НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютеризовані системи управління»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Розпроділена система автоматизації та управління між сервісним музикальним контентом користувачів»

Виконав (-ла):
студент (-ка) IV курсу, групи ІА-62
Богомол Роман Олександрович _____

Керівник:
Асистент
Вовк Євгеній Андрійович _____
_____

Рецензент:
Доцент АСОІУ, к.т.н. Сперкач Майя Олегівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютеризовані системи управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Олександр РОЛІК

«_»_____ 20__р.

ЗАВДАННЯ

на дипломний проєкт студенту

Богомола Романа Олександровича

1. Тема проєкту «Розпроділена система автоматизації та управління між сервісним музикальним контентом користувачів», керівник проєкту Вовк Євгеній Андрійович асистент, затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту

4. Зміст пояснювальної записки Вступ, огляд існуючих рішень, огляд комплексного рішення, огляд онлайн рішення для часткового аналізу, формулювання технічного завдання, технічне рішення, архітектура додатку, технологія реалізації програмне рішення, архтектура програмного рішення, діаграма компонентів, компонент серверу, рівень комутації, опис клієнта, виконана програма.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) UML-діаграма, контекстна блок-схема алгоритму роботи системи, структурна схема.

6. Дата видачі завдання ___

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримати завдання на ДР	13 квітня 2020р.	
2	Аналіз предметної області	14 квітня 2020р.	
3	Огляд існуючих рішень	15 квітня 2020р.	
4	Пошук архітектурного рішення	16 квітня 2020р.	
5	Пошук технічного рішення	20-22 квітня 2020р.	
6	Розробка серверного додатка системи	22-31 квітня 2020р.	
7	Розробка взаємодії серверного додатка з джерелами даних	31 квітня 2020р.	
8	Реалізація концепції 'Rest API' і взаємодія клієнтської частини з серверним додатком	1-5 травня 2020р.	
9	Реалізація інтерфейса користувача	5-10 травня 2020р.	
10	Тестування та налаштування міжсервісной взаємодії	10-15 травня 2020р.	
11	Опис роботи	15-21 травня 2020р.	
12	Оформлення текстової та графічної документації в межах формування звіту	1-10 червня 2020р	

Студент _____ Р.О Богомол
(підпис) (ініціали, прізвище)

Керівник роботи _____ Є. А. Вовк
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Богомол Р.О. Розпроділена система автоматизації та управління між сервісним музикальним контентом користувачів. КПІ ім. Ігоря Сікорського, Київ, 2020.

Проект містить 67 с. тексту, 44 рисунків, 32 літературних джерел та 6 додатків.

Ключові слова: управління музичними сервісами, клієнт-серверна архітектура, сервісно-орієнтована архітектура.

Об’єктом дослідження є музичний сервіс.

Предметом дослідження зовнішні API музичних сервісів.

Дипломний проєкт призначений для розробки програмного продукту, який реалізовуватиме завдання централізованого управління і агригації музичних сервісів. Архітектурне рішення по вирішенню даного завдання складатиметься із клієнт-серверної архітектури, серверна частина якої будуватиметься на основі мікросервісів.

SUMMARY

ANNOTATION

Bogomol RO Distributed system of automation and management between service music content of users. KPI them. Igor Sikorsky, Kyiv, 2020.

The project contains 67 pages. text, 44 drawings, _ literary sources and 6 appendices.

Keywords: music services management, client-server architecture, service-oriented architecture.

The object of research is the music service.

The subject of research is external API music services.

The diploma project is designed to develop a software product that will implement the tasks of centralized management and aggregation of music services. The architectural solution to solve this problem will consist of a client-server architecture, the server part of which will be built on the basis of microservices.

Пояснювальна записка
до дипломного проєкту
на тему: «Система автоматизованого тестування»

Київ – 2020 року

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІА62.030БАК.005 ПЗ	Пояснювальна записка	67		
6	А3	ІА62.030БАК.005 Э1	Структурна схема	1		
7	А3	ІА62.030БАК.005 Д1	Діаграма класів механізму	1		
8	А3	ІА62.030БАК.005 Д2	Контекстна блок-схема	1		
9			роботи			
10	А3	ІА62.030БАК.005 Д3	Діаграма взаємодії із	1		
11			системою			
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІА62.050БАК.005 ТП		
Розроб.		Богомол Р.О.			Розпроділена система автоматизації та управління між сервісним музикальним контентом користувача	Літ.	Лист	Листів
Перевір.		Вовк Є.А.				Т	1	1
Реценз.						НТУУ «КПІ» ФІОТ Група ІА-62		
Н. Контр.								
Затв.								

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API - application programming interface

ІТС – інформаційно-телекомунікаційні системи

ПЗ – програмне забезпечення

XML - eXtensible Markup Language

JSON - JavaScript Object Notation

БД – база даних

UTF-8- Unicode Transformation Format

SOA - service-oriented architecture

XAML - eXtensible Application Markup Language

ВСТУП

В сучасну еру розвитку комп’ютеризованих технологій можливість прослуховування музики стає все більш зручнішим. Це все можна спостерігати, аналізуючи еволюцію зберігання і програвання аудіо - даних, виділяючи такі етапи розвитку: превалювання фізичних носіїв для запису файлів електронного формату на кінцевій машині користувача та етап розповсюдження онлайн - доступу.

Компакт - диски мають свої недоліки головними з яких можна назвати нестійкість до пошкоджень, необхідність зберігання даних, незручність використання, а також при пошкоджені носія можлива втрата даних. До того ж, через лімітованість ресурсів інтернету, протягом довгого часу зберігання музичних списків відтворення відбувались на машині користувача.. Як результат, це викликало низку недоліків, які пов'язуються із неможливістю своєчасно отримувати новинки музики в момент їх виходу та появи на ринку, викликало труднощі при пошуку музичних композицій. Всі перераховані недоліки започаткували сервіси для відтворення музичних списків онлайн. Дані сервіси, об'єднують велику кількість спільних ознак, а також виділяються своїми особистими перевагами чи недоліками. Вони є зручними для користувача, але через розвиток ресурсів такого типу та значним зростанням їхньої кількості, постало завдання забезпечення централізованого доступу і агрегації сервісів.

Завданням дипломного проєкту було проаналізувати готові рішення по вирішенню задачі централізованого управління та агрегації музичних сервісів, оглянути основні переваги, а також виділити недоліки аналогів, спроектувати архітектуру рішення і на її основі реалізувати програмний продукт системи автоматизації та управління міжсервісним музикальним контентом користувача в межах оптимальних технологій розробки систем даного типу.

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд комплексного рішення

В рамках дипломного проекту, для визначення шляхів вирішення завдань агрегації і централізованого доступу до музичних платформ було розглянуто існуючі сервіси на прикладі таких як Deezer, YouTube Music, iTunes, Spotify, Apple Music, які на даний момент є доступними в країнах Північної та Південної Америк, Австралії, Нової Зеландії в більшості країн Європи, а також в країнах Азії. Дані сервіси доступні для більшості сучасних платформ, включаючи Windows, MacOS, Linux, а також IOS та Android, будучи адаптованими в першу чергу до смартфонів і планшетів. Аналізуючи вище описані сервіси можна виділити ряд спільних ознак для кожного із них:

- музика може бути переглянута або відсортована (знайдена) завдяки можливості пошуку з використанням різних параметрів, таких як виконавець, альбом, жанр, список відтворення або звукозаписний лейбл;
- користувачі можуть створювати, редагувати і обмінюватися списками програвання, ділитися треками в соціальних мережах, а також створювати спільні списки з іншими користувачами;
- доступна можливість прослуховування музики в режиму офлайн;
- користувачі можуть підписуватись на виконавця, групу чи на певний жанр і при виході новинок музики клієнт отримає сповіщення;
- доступна можливість формувати список улюблених треків відносно жанрів яких буде формуватись запропоновані виконавці чи альбоми.

Також деякі сервіси містят у собі певні особливості, які відрізняють їх від своїх аналогів. Кожна із платформ має різні умови підписки, а також накладання певних обмежень на користувача. Доприклад Apple Music включає можливість перегляду тексту слів музики чого не мають багато інших сервісів. Також деякі сервіси є недоступними для всіх країн.

Аналізуючи функціонал описаних вище сервісів: iTunes, Spotify, Deezer YouTube Music можна виділити ряд важливих загальних функцій, які працюють у кожному із сервісів за схожим алгоритмом:

- пошук аудіоконтенту;
- створення список відтворення;
- збереження музики для прослуховування в режимі офлайн.

Для агрегації і централізованого управління музичних сервісів слід реалізувати продукт, який буде містити в собі загальні ознак і функцій, які притаманні кожному із вищезгаданих сервісів і які відіграють основну роль у роботі над музичним контентом.

1.2 Огляд онлайн рішення для часткового аналізу

Під час огляду і аналізу існуючих рішень було знайдено наступні додатки: soundiiz[1] і musconv[2].

Додаток soundiiz реалізований за допомогою веб-орієнтованої архітектури, об’єднує близько тридцяти музичних сервісів і реалізовує алгоритми передачі музичних даних між потоковими платформами. До основних можливостей даного додатку можна віднести синхронізацію списків відтворення за допомогою якої можна централізовано керувати створенням, видаленням, редагуванням музичного контенту всіх сервісів.

Souniiz має зручний інтерфейс користувача де можна з легкістю переглядати всю доступну інформацію у кожному із сервісів рис 1.1

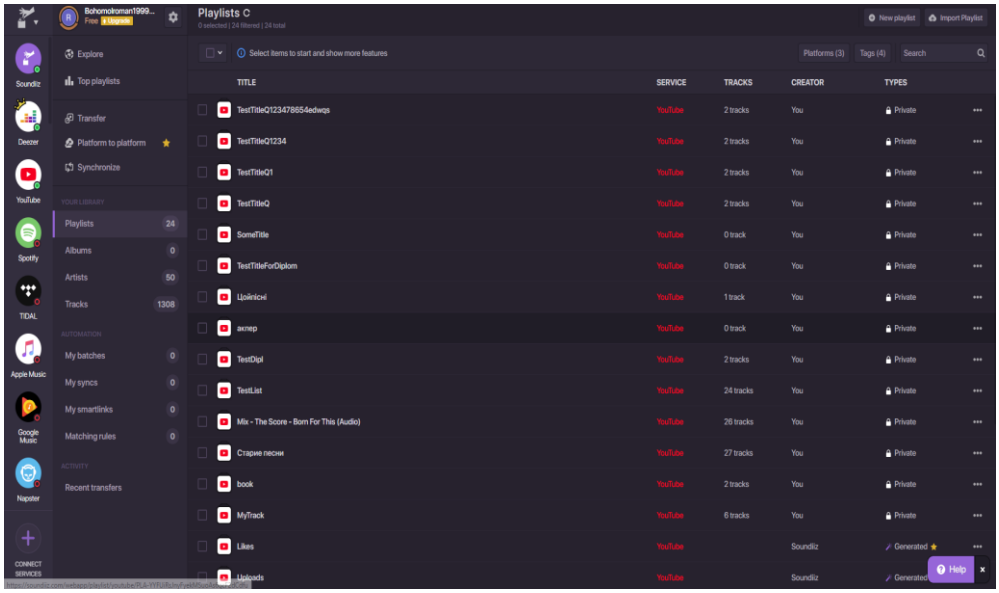


Рисунок 1.1 – Вигляд основного вікна додатка Souniiz

Також реалізована можливість імпорту та експорту даних за допомогою файлів з розширенням M3U, XSPF, TXT, CSV, URL посилання та ін. рис 1.2.

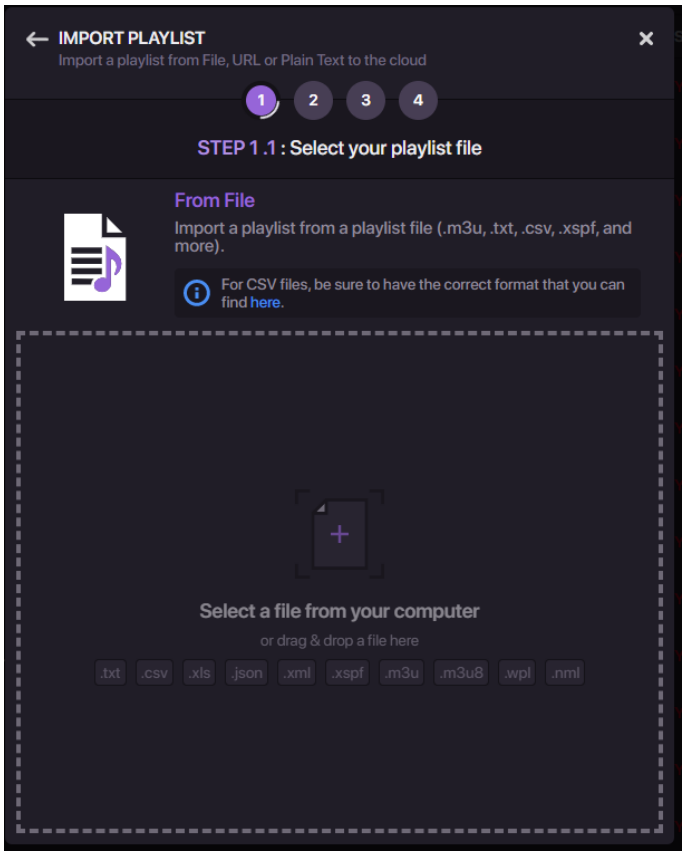


Рисунок 1.2 – Вигляд вікна імпорту даних за допомогою файлів.

Для авторизації у кожному із сервісів виділяють своє графічне вікно користувача, наприклад для YouTube Music воно виглядає як реєстрація для всіх інших сервісів Google.

Даний менеджер вирішує основні проблеми, які пов’язані із агрегацією сервісів і централізованим управлінням, реалізовує можливість передачі аудіо контенту між платформами, але також містить суттєву кількість недоліків, головна з яких це платна підписка для користувача і накладання значної кількості обмежень для клієнтів, які використовують додаток безплатно, також відсутня можливость завантаження аудіо треків і прослуховування.

MusConv – платформа яка об’єднує в собі більш ніж 30 музичних сервісів, має багато спільних можливостей із Soundiiz, наприклад синхронізацію музичних списків, експорт і імпорт даних. Додаток має і суттєву перевагу над Soundiiz, тому що має свою віконну графічну версію користувача(десктоп) рис 1.3 і є доступним для більшості сучасних пристроїв

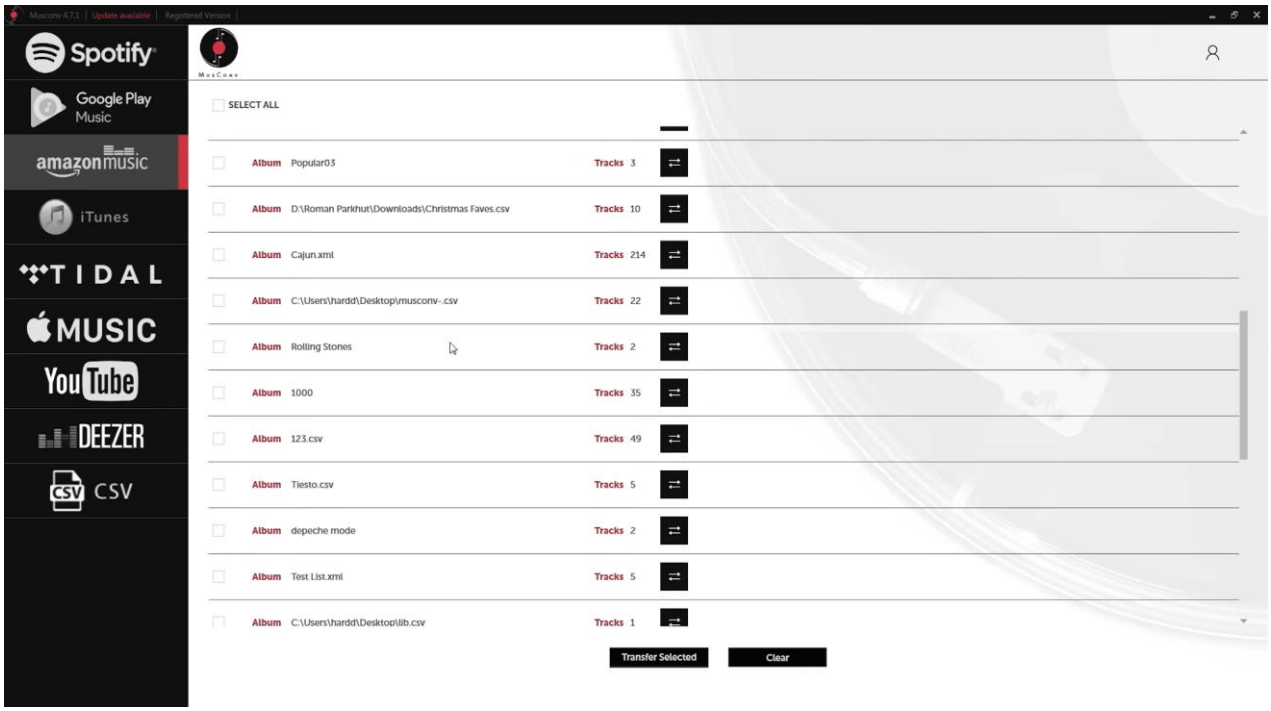


Рисунок 1.3 – Головне меню віконного графічного додатка користувача

Також присутня комфортна можливість програмної обробки файлів (парсинг) з розширенням csv за допомогою якого процес створення списків відтворення спрощується.

Серед переваг даних рішень можна виділити зручність створення нових списків відтворення, адже додається можливість шаблонного завантаження файлів у вказаному форматі і їх програмна обробка (парсинг). Ці файли створюються за допомогою популярних музичних плеєрів, що дозволяють користувачеві

відмовитись від ручного вводу інформації, а також надає можливість парсингу даних за допомогою URL-посилання. Серед недоліків можна виділити платну підписку для кожного із продуктів, відсутність віконного додатку під ОС Windows, Linux у Soundiiz, який міг би спростити доступ користувачів до сервісу. Також при великому навантажені на дані ресурси можливе «зависання» програми на деякий час, для того аби не призвести даних незручностей для користувача накладаються обмеження по кількості створених треків в одному із списку відтворень.

1.3. Формулювання технічного завдання

Після аналізу і огляду існуючих аналогів по централізованому управлінню і агрегацією музичних сервісів було виявлено ряд недоліків:

- платна підписка для кожного із сервісів, яка обумовдює значну кількість накладених обмежень для користувачів;
- відсутність графічного відображення вікна користувача у платформі Soundiiz;
- при великому навантаженню чи обробці значної кількості даних можливе «зависання» роботи додатку.

Також було виявлено список переваг даних сервісів:

- можливість шаблонного завантаження файлів і їхня програмна обробка;
- створення списку відтворення за допомогою URL – посилання;
- присутня функція ручного введення просто тексту за допомогою якого буде знайдено відповідний трек і добавлено до сервісу.

Проаналізувавши описані вище тези відповідно до виявленого огляду було взято за мету спроектувати шаблонне рішення системи і на основі якого реалізувати програмний продукт, який буде поєднувати всі переваги, а також недопускатиме вищеописаних недоліків.

2. ТЕХНІЧНЕ РІШЕННЯ

2.1 Архітектура додатку

Для того щоб створити комплексне програмне рішення в межах описаної в технічному рішенні задачі, необхідно визначити загальний тип міжкомпонентної взаємодії в межах побудови архітектурного рішення, виходячи з критерію про наявність декількох пристроїв в одного користувача та значної кількості кінцевих користувачів сервісу, залишаються наступні архітектурні рішення: клієнт-серверна архітектура [3], Сервіс-орієнтована архітектура (SOA, англ. Service-oriented architecture)[4].

Клієнт-серверна архітектура рис 2.1 – архітектура комп’ютерної мережі в якій динамічна множина клієнтів (віддалених процесорів) отримують послугу від централізованого сервера (хост-комп’ютера). Клієнтські комп’ютери надають інтерфейс для користувача за допомогою якого комп’ютер формує коректний запит до сервера і відображатиме відповідь сервера. Сервери перебувають в режимі очікування запитів від клієнта, а далі обробляють їх і повертають відповіді. Сервери забезпечують прозорий стандартизований інтерфейс для клієнтів, щоб ті не мали жодної інформації щодо специфіки системи (тобто апаратного та програмного забезпечення), що надає послугу. Клієнти часто знаходяться на робочих станціях або на персональних комп’ютерах, а сервери розташовані в інших місцях мережі, як правило, працюють на більш потужних машинах. Ця обчислювальна модель особливо ефективна, коли клієнти та сервер мають різні завдання, які вони регулярно виконують. Наприклад, в обробці даних в лікарні, на клієнтському комп'ютері запущена графічна віконна програма для введення інформації про пацієнтів, а на серверному комп'ютері працює інша програма, яка управляє базою даних, в якій інформація постійно зберігається. Багато клієнтів можуть отримати доступ до інформації сервера одночасно, і в той же час

клієнтський комп'ютер може виконувати інші завдання, наприклад, надсилання електронної пошти.

Проаналізувавши даний архітектурний підхід можна виділити наступні компоненти для реалізації клієнт-серверної взаємодії:

- набір серверів, які обробляють запити клієнтів;
- набір клієнтів, що звертаються до серверної частини системи;
- рівень комунікації, що базується на мережевих протоколах взаємодії та забезпечує обмін інформацією між компонентами системи, а саме клієнтами та серверами.

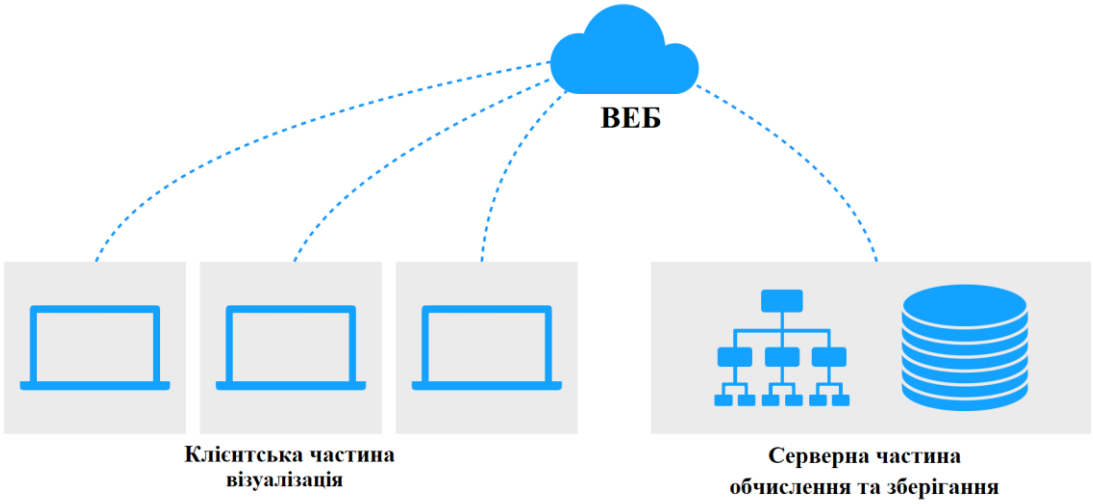


Рисунок 2.1 – Клієнт – серверна архітектура

Проаналізувавши дане архітектурне рішення можна винести ряд переваг роботи системи в межах клієнт-серверної взаємодії:

- не потрібно дубувати програмний код роботи сервера для кожного із клієнтів;
- відносно не велике навантаження на машину користувача так як більшість обчислень відбуватиметься на сервері;
- більшість даних користувача зберігають на сервері, який має більший захист чим клієнтські машини.

Також можна винести ряд суттєвих недоліків:

- вихід із ладу централізованого сервера може призвести до непрацездатності всією обчислюваної мережі;

- при великій кількості запитів до сервера можливе уповільнення роботи мережі;
- підтримка даної системи потребує професійного фахівця;
- висока вартість обладнання.

Також даний архітектурний підхід дозволяє розділити логіку по обробці і зберіганню даних на декілька окремих серверів, що і є основою для побудови багаторівневої архітектури. Це дозволить збільшити швидкість обробки запиту клієнта розділивши функції обробки, зберігання і представлення даних рис 2.2.

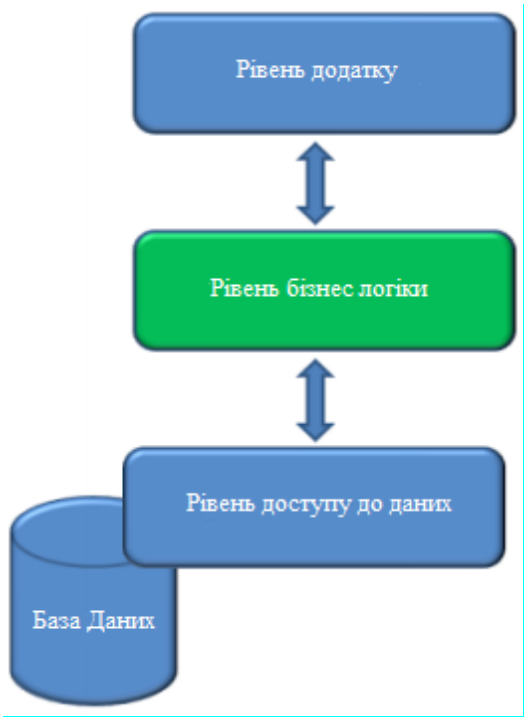


Рисунок 2.2 Приклад тришарової архітектури.

Для реалізації програмного рішення в межах серверної частини клієнт – сервреної – взаїмодії слід розглянути архітектурний підхід SOA.

Сервісно-орієнтована архітектура(SOA) є парадигмою організації та використання розподілених можливостей, які можуть перебувати під контролем різних доменів власності. Це стиль розробки програмного забезпечення, коли послуги надаються іншим компонентам компонентами додатків, через стандартизований протокол зв'язку по мережі рис. 2.3.

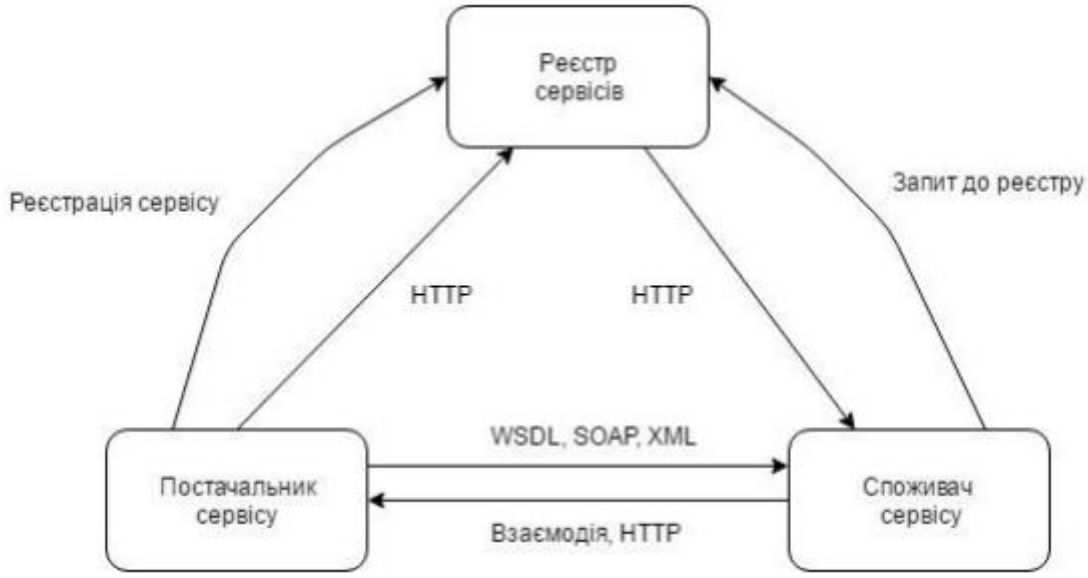


Рисунок 2.3 Базова структура SOA

Системи, які реалізовані на основі даного архітектурного підходу взаємодіють зазвичай за допомогою базового стандартизованого протоколу обміну даними SOAP або архітектурного ситилю REST, хоча є й інші реалізації (jinni, CORBA)[5].

REST – це стиль архітектури програмного забезпечення на основі якого будується клієнт серверна взаємодія. Даний стиль, як правило, використовується для побудови веб – служб. Кожна одиниця інформації є доступною за допомогою URL – посилання.Кожне посилання має статичний формат. Був розроблений в дисертації Роя Філдінга в 200 році, як альтернатива SOAP, коли запит клієнта несе в собі вичерпну інформацій про бажане відповіді сервера і сервер не зобов'язаний зберігати сесію взаємодії з клієнтом.

Стандартні методи такі:

- GET – найбільш популярний метод, який слугує для повернення даних і не змінює їх;
- POST - даний метод використовують для вставки нових записів;
- PUT – даний метод реалізовує зміну записів;
- PATCH – даний метод використовують для зміни ідентифікатора вже існуючого запису;
- DELETE – даний метод використовують для видалення записів.

Аналізуючи даний архітектурний стиль можна виділити наступні переуваги:

- гнучкість і вміння пристосовуватись до інших умов;
- простота редагування і внесення змін;
- масштабованість;
- портативність компонентів;
- надійність.

Однією із варіантів SOA рхітектури являється мікросервісна архітектура [6].

Архітектурний стиль мікросервісів – це підхід, коли система будується на основі двох і більше мікросервісів, які є незалежними одним від одного і спілкуються між собою за допомогою механізмів gRPC, HTTP, AMQP [7][8][9] рис 2.4.

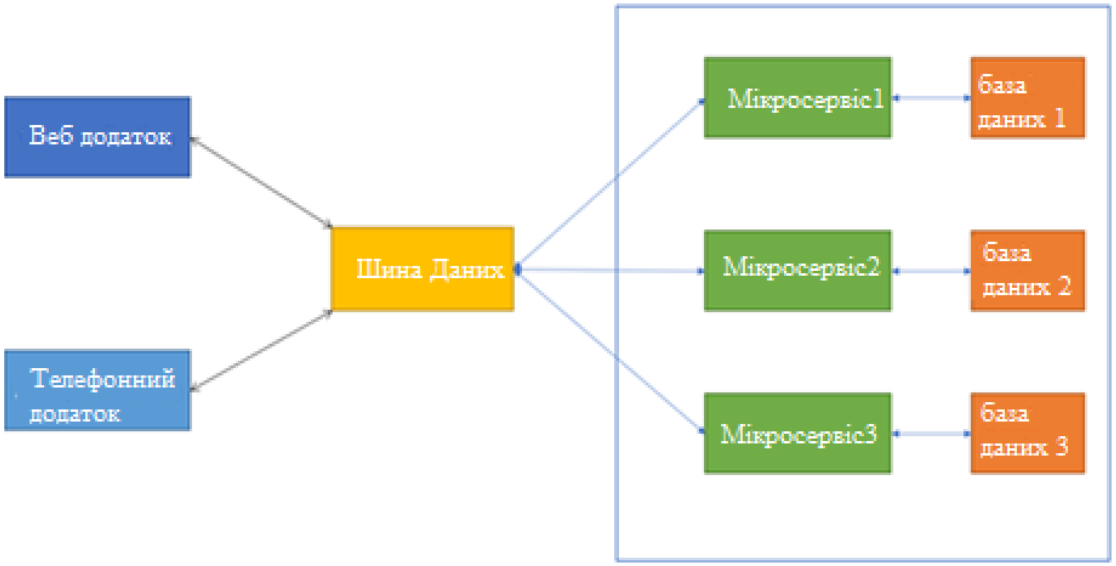


Рисунок 2.4 – Приклад мікросервісної архітектури

HTTP (протокол передачі гіпертексту) - це протокол програми для розподілених, спільних, інформаційних систем гіпермедіа, який дозволяє користувачам обмінюватися даними у всесвітній павутині. HTTP був винайдений поряд із HTML для створення першого інтерактивного текстового веб-браузера. На даний момент протокол залишається одним із головних механізмів користування інтернетом. Як тільки веб-користувач відкриває свій веб-браузер, користувач опосередковано використовує HTTP.

HyperText Transfer Protocol Secure (HTTPS) - це зашифрована версія HTTP, яка є основним протоколом, що використовується для передачі даних по всесвітній павутині.

HTTPS захищає зв’язок між вашим веб-переглядачем та сервером від перехоплення та зловмисника. Це забезпечує конфіденційність, цілісність та автентифікацію для більшості сьогоднішніх трафіків WWW

Кожен мікросервіс реалізовується в межах своїх бізнес-потреб і виконує якийсь конкретний бізнес-процес та розгортається незалежно від інших мікросервісів за допомогою автоматизованого середовища. Кожен сервіс є незалежним один від одного і може реалізовуватись за допомогою різних технологій. Даний підхід має наступний список переваг:

- незалежне розгортання, розробка, маштабування;
- простота заміни однієї реалізації сервісу іншою;
- ефективне використання ресурсів;
- легкість доповнення функціоналу;
- відносно невелика кодова база в межах конкретного процесу дозволяє швидко адаптовуватись новим розробникам;
- незалежність системи від мікросервісів: вихід із ладу одного мікросервісу в більшості випадків не призводить до виходу із ладу інших мікросервісів.

Дана архітектура як і всі інші не являється ідеальною і має певні недоліки:

- значні накладні витрати на інфраструктуру, моніторинг і операційні дії;
- даний архітектурний підхід наслідує усі проблеми розподілених систем;
- обмеження і рамках «одна команда — один сервіс» викликає проблеми повязані із відсутністю функціоналу, який розробляється паралельно іншою командою.
- незажність кожного із сервісів призводить до проблеми дублювання коду;
- ускладнене налагодження;

- проблеми із мережевими затримками в рамках міжсервісної взаємодії;
- відносно важкий процес розгортання, тестування і забезпечення безпеки.

Проаналізувавши даний архітектурний підхід можна зробити висновок, що мікросервіси – це архітектурне рішення для розробки програмного продукту, який складається з набору сервісів, які взаємодіють між собою за допомогою API [10]. Переваги даного підходу полягають в можливості реалізації кожного із сервісів за допомогою різних технологій зберігання даних і різних мов програмування.

Також слід виділити те, що дана архітектура забезпечує можливість маштабування. В галузі програмного забезпечення маштабованістю можна назвати певною властивістю системи, процесу чи мережі, яка свідчитиме про можливість системи бути легко розширеною або обробити більшу кількість інформації. В межах мікросервісної архітектури маштабованістю можна назвати здатність підтримувати велику кількість архітектурних компонентів (мікросервісів), та комунікацію між ними.

Маштабованість поділяють на дві широкі категорії:

- горизонтальне маштабування;
- вертикальне маштабування.

Горизонтальне маштабування (вихід / вхід) означає додавання більшої кількості вузлів до (або видалення вузлів з) системи, наприклад додавання нового комп'ютера до розподіленого програмного забезпечення. Приклад може включати масштабування з одного веб-сервера на три. Високопродуктивні обчислювальні програми, такі як сейсмічний аналіз та біотехнологічні навантаження, масштабуються горизонтально для підтримки завдань, які колись потребували б дорогих суперкомп'ютерів. Інші робочі навантаження, наприклад, великі соціальні мережі, перевищують потужність найбільшого суперкомп'ютера, і їх можна обробляти лише масштабованими системами. Використання цієї масштабованості вимагає програмного забезпечення для ефективного управління ресурсами та їх обслуговування.

Вертикальне маштабування або маштабування вгору означає додавання ресурсів до (або вилучення ресурсів з) одного вузла, як правило, включає додавання процесора, пам'яті або зберігання до одного комп’ютера.

У відповідності до специфіки і обмежень, переваг та недоліків кожного з видів архітектурних підходів для оптимальної реалізації завдання централізованого управління і агрегації музичних сервісів було спроектовано загальну схему взаємодії системи, яка складатиметься із трьох компонентів:клієнт, сервер, зовнішні API(для взаємодії із музичними сервісами) рис 2.5.

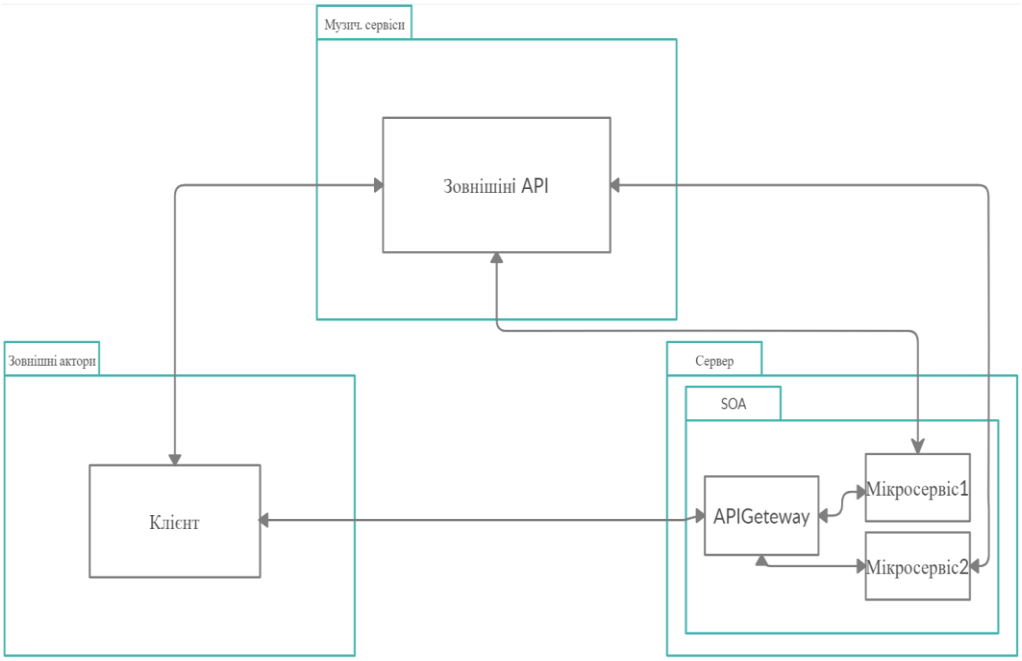


Рисунок 2.5 – Схема роботи системи

Було прийняте рішення про забезпечення взаємодії клієнтського додатку з центральним сервісом системи на основі клієнт-серверної архітектури. Виходячи з критерію завантаженості класу подібних підсистем серверне рішення по агрегації контенту та його отриманню було спроектоване на основі мікросервісної архітектури, що забезпечує можливість горизонтального та вертикального масштабування кінцевого рішення у відповідності до навантаження в поточний момент часу.

Взаємодія в межах мікросервісного рішення побудована з використанням шини даних, що забезпечує централізоване управління міжсервісним трафіком та гарантує його отримання екземпляром конкретного мікросервісу, також шина даних

є вхідною точкою в механізм міжсервісної комунікації та відповідає за побудову маршрутів для мережевих пакетів

2.3 Аспекти та базові механізми побудови рішення в межах даного класу підсистем

Для програмної взаємодії із кожним музичним сервісом було проаналозовано і оглянуто документацію по роботі з API (англ. Application programming interface).

Інтерфейс програмування додатків (API) - це інтерфейс, який визначає взаємодію між декількома програмними посередниками. Він визначає види запитів, які можна здійснювати, як їх здійснювати, формати даних, які слід використовувати, конвенції, які слід дотримуватися тощо. Він також може забезпечити механізми розширення, щоб користувачі могли розширювати існуючу функціональність різними способами та в різній мірі. API може бути розроблений на основі галузевого стандарту для забезпечення сумісності.

Аналізуючи API кожного із сервісів можна зробити висновок, що кожна платформа, для отримання обмежених прав доступу до даних користувача, реалізує загальним тип авторизації на основі протоколу OAuth2.0[11].

OAuth 2.0 - протокол авторизації, що дозволяє видати одному сервісу (з додатком) права на доступ до ресурсів користувача на іншому сервісі. Протокол позбавляє від необхідності довіряти додатком логін і пароль, а також дозволяє видавати обмежений набір прав, а не все відразу.

Як і перша версія, OAuth 2.0 заснований на використанні базових веб-технологій: HTTP-запитів, перенаправлення і тд. Тому використання OAuth можливо на будь-якій платформі з доступом до інтернету і браузеру: на сайтах, в мобільних і desktop-додатках.

Ключова відмінність від OAuth 1.0 - простота. У новій версії немає громіздких схем підпису, скорочено кількість запитів, необхідних для авторизації.

Загальна схема роботи програми, що використовує OAuth, така:

- отримання авторизації;

- звернення до захищених ресурсів.

Результатом авторизації є – маркер доступу(access token[12]) (зазвичай просто набір символів), представлення якого є пропуском до захищених ресурсів. Звернення до них в найпростішому випадку відбувається по HTTPS із зазначенням в заголовках або в якості одного з параметрів.

Маркер доступу - це те, що програми використовують для подання запитів API від імені користувача. Маркер доступу представляє авторизацію конкретної програми для доступу до певних частин даних користувача.

Маркери доступу повинні зберігатись у таємниці під час транзиту та зберігання даних. Єдиними сторонами, які повинні бачити маркер доступу, є сама програма, сервер авторизації та сервер ресурсів. Додаток повинен гарантувати, що зберігання маркера доступу не буде доступним для інших програм на тому ж пристрої. Маркер доступу може використовуватися лише через https-з'єднання, оскільки передача його через незашифрований канал зробить тривіальним для перехоплення третіх сторін.

У протоколі описано кілька варіантів авторизації, що підходять для різних ситуацій:

- авторизація для повністю клієнтських додатків (мобільні і desktop-додатки);
- відновлення попередньої авторизації;
- авторизація за логіном і паролем;
- авторизація для додатків, що мають серверну частину (найчастіше, це сайти і веб-додатки).

Розглянемо деякі варіанти авторизації.

Авторизація для додатка, якаий має серверну частину виглядатиме наступним чином рис 2.6

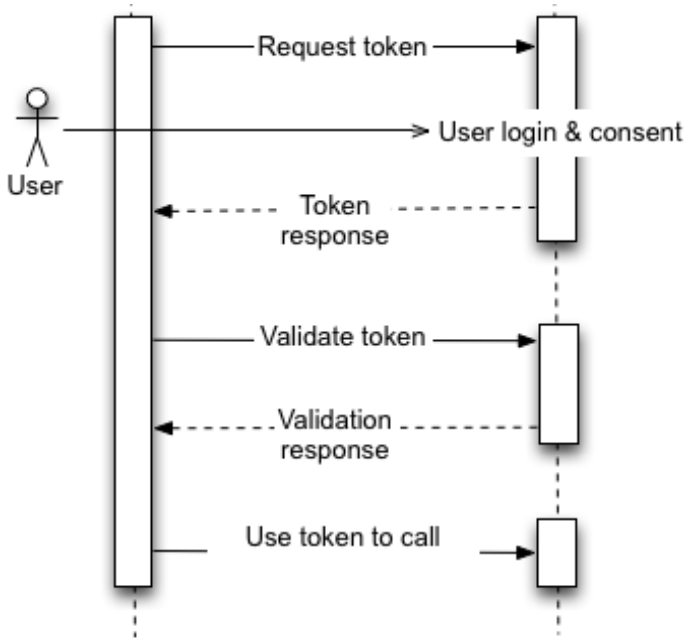


Рисунок 2.6 – Схема авторизації для додатка із серверною частиною.

Даний тип авторизації працює за допомогою наступного алгоритму:

- перенаправлення на сторінку авторизації;
- на сторінці авторизації у користувача запитується підтвердження видачі прав;
- у разі згоди користувача, браузер перенаправляє на URL, вказаний при відкритті сторінки авторизації, з додаванням в GET-параметри спеціального ключа - authorization code;
- сервер додатка виконує POST-запит з отриманим authorization code як параметр. В результаті цього запиту повертається access token.

Це найскладніший варіант авторизації, але тільки він дозволяє сервісу однозначно встановити додаток, що звертається за авторизацією (це відбувається при комунікації між серверами на останньому кроці). У всіх інших випадках авторизація відбувається повністю на стороні клієнта і зі зрозумілих причин можлива маскування однієї програми під іншу. Це варто враховувати при впровадженні OAuth-аутентифікації в API сервісів.

Розглянемо авторазицію за допомогою клієнтських додатків рис 2.7

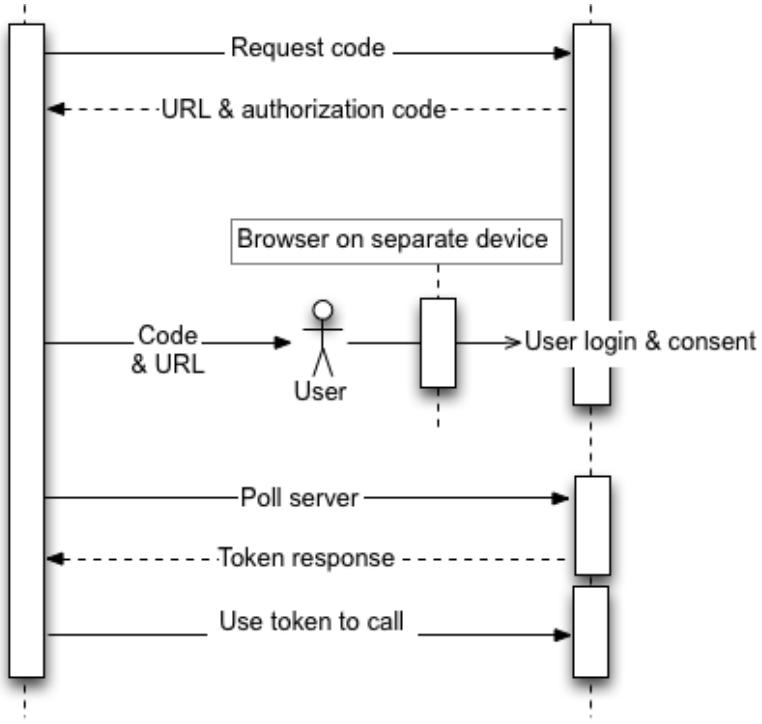


Рисунок 2.7 – Схема авторизації на стороні клієнта

Алгоритм роботи для даного рішення працю наступним чином:

- відкриття вбудованого браузера зі сторінкою авторизації;
- у користувача запитується підтвердження видачі прав;
- у разі згоди користувача, браузер перенаправляє на стандартну сторінку у фрагменті (після #) URL якої додається маркер доступу;
- додаток перехоплює перенаправлення і отримує маркер доступу з адреси сторінки.

Перевага даного рішення полягає у тому, що тут не потрібно серверної частини і додаткового виклику сервер – сервер для обміну кода авторазації на маркер доступу. Проте даний підхід вимагає реалізації на клієнській частині вікна браузера, який міг би відображати стандартне вікно авторизації для введення логіна і пароля, а також надання клієнтом доступу до своїх даних.

Враховуючи важкість реалізації даного протоколу на стороні сервера було прийнято рішення реалізації авторизації на основі протоколу OAuth2.0 на основі клієнтських додатків.

2.4 Технологія реалізації

Для реалізації спроектованих архітектуриних рішень було проаналізовано наявні технології, на основі яких можна розробити програмний продукт в межах обумовленої архітектури. Було розглянуто перелік технологій, що надають інструментарій по реалізації міжклієнтської взаємодії в межах клієнт – серверної архітектури програмного рішення такі як JAVA, .NET. На основі порівняння технологій[13] WPF і Swing, а саме на основі характеристик ресурсозатратності по швидкодії, оперативній пам’яті, та складності реалізації. Результати порівняння продемонстровано на рис 2.8.

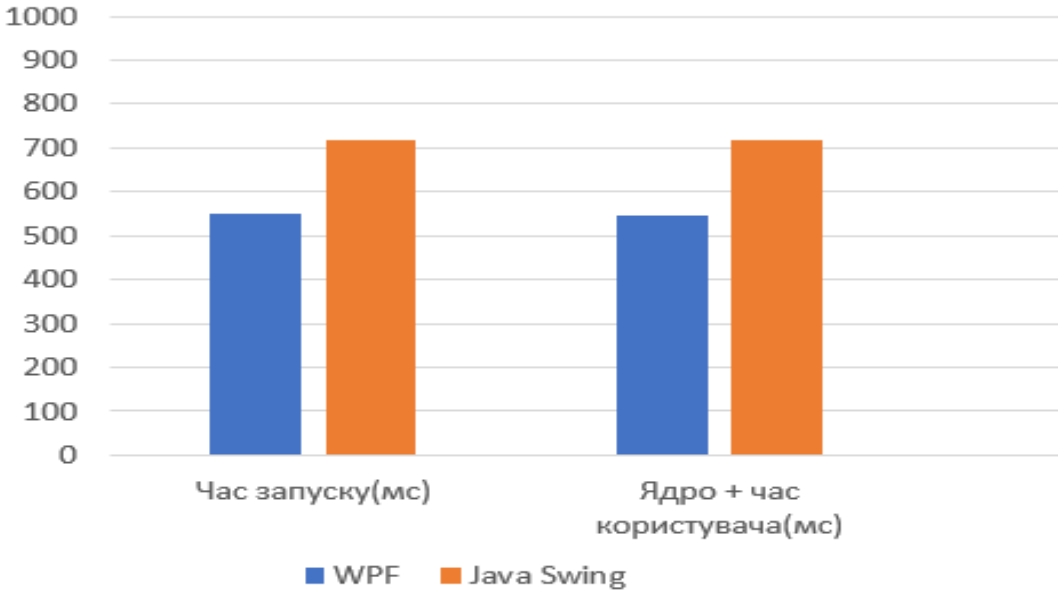


Рисунок 2.8 – Порівняння ресурсозатратності по швидкодії WPF та Swing.

В результаті аналізу та огляду проведених вище порівнянь було прийнято рішення для розробки клієнтської частини системи обрати платформу .NET.

.NET - це програма розробки програмного забезпечення - супутня екосистема інструментів, мов та режимів виконання - створена корпорацією Майкрософт для полегшення розробки програм на різних платформах - від настільних комп’ютерів до мобільних пристроїв. Хоча .NET (вимовляється крапка в мережі, а іноді і записується як .Net. спочатку був пов'язаний із власною операційною системою та платформами Microsoft під час запуску на початку 2000-ниї років, .NET-програми тепер можна писати для Інтернету, iOS, Android, Linux, MacOS та багато іншого - і .NET - це формальний стандарт і офіційно доступний як відкритий код.

Microsoft описує .NET як "послідовне об'єктно-орієнтоване середовище програмування, незалежно від того, зберігається чи виконується об'єктний код локально, виконується локально, але розповсюджується в Інтернеті, або виконується віддалено". .NET має на меті забезпечити безпечне виконання коду, забезпечити кращу продуктивність, ніж інтерпретовані мови, та забезпечити послідовність роботи розробників у широкому спектрі програм.

.NET Framework існував майже 20 років і зазнав багатьох змін, і компоненти згодом застаріли. На даний момент у .NET є три основні шари:

- стандартна бібліотека .NET включає компоненти, які формуватимуть інфраструктуру майже для будь-якої програми, яку ви пишете - класи та типи, які допомагають виконувати повсякденні завдання, такі як обробка струн та примітивів, створення підключень до бази даних, виконання вводу / виводу операції тощо;
- необов’язкові моделі додатків містять код програмний код для різних платформ, на яких ви можете розгорнути свою .NET-програму Існує ряд моделей Windows, а також для інших платформ: наприклад, ASP.NET для веб-додатків, а також моделі для Mac і різних мобільні платформи;
- загальна інфраструктура - це базовий рівень компонентів, які дозволяють практично виконувати всю екосистему на практиці, від компіляторів до мов і компонентів часу виконання.

Основні компоненти .NET Framework працюють разом, щоб упростити процес написання додатків. Стандартні моделі бібліотеки надають багато коду для вирішення основних завдань програмування. І загальна інфраструктура забезпечує значну частину роботи з розгортання цих програм.

Код, написаний будь-якою з мов .NET, збирається на мову проміжного байт-коду, яку називають загальною проміжною мовою, або CIL. Код CIL не є читабельним для людини, але може переноситися через операційні системи та платформи. Потім CIL знову компілюється загальною мовою виконання або CLR. Реалізації CLR залежать від платформи, і вони компілюють код CIL у машинний

код, який може бути виконаний на платформі в даний момент. Різні версії CLR підтримують своєчасну компіляцію,

У процесі створення локального машиночитаного коду CLR також управляє безліччю функціональних додатків низького рівня, таких як збирання сміття(garbage collection) та багатопоточність, що є вирішальним для продуктивності додатків, але часто стомлює розробників. Спільно CIL та CLR складають загальну мовну інфраструктуру .NET.

Аналізуючи описані вище тези щодо платформи .NET можна виділити ряд наступних переваг:

- забезпечення узгодженої об'єктно-орієнтованого середовища програмування для локального збереження і виконання об'єктного коду, для локального виконання коду, розподіленого в інтернеті, або для віддаленого виконання;
- забезпечення середовища виконання коду, що мінімізує конфлікти при розгортанні програмного забезпечення та управлінні версіями;
- забезпечення середовища виконання коду, що виключає проблеми з продуктивністю середовищ виконання сценаріїв або інтерпретованого коду;
- забезпечення єдиних принципів розробки для різних типів додатків, таких як додатки Windows і веб-додатки.

В сучасному IT світі в рамках платформи .NET виділяють найчастіше використованою мову програмування C#. Вона була розроблена корпорацією Microsoft як частина ініціативи .NET, і більшість класів у стандартній бібліотеці .NET написані на C #. Мова об'єктно-орієнтована і розроблена таким чином, щоб вона була досить схожою на C, щоб було легко для розробників C, C ++, Java та JavaScript для швидкого вивчення та використання.

В даний час Microsoft також використовує дві інші мови програмування, які можна використовувати для запису .NET Framework. Перша це - F #, функціональна мова програмування, яка є частиною сімейства мов ML, яка в кінцевому підсумку має коріння з LISP; друга - Visual Basic, поважна, легка в засвоєнні мова

програмування Microsoft для розробки програм клієнт-сервер. Оскільки .NET складається з відкритих стандартів, кожен може написати мову, яка компілюється в байт-код CIL і може бути виконана CLR. У Вікіпедії є список більш ніж 20 мовних проектів CLI, які зараз підтримуються. Майже всі вони представляють .NET-порти існуючих мов, від Pascal до JavaScript і навіть COBOL.

Те, що таке різноманіття мов може співіснувати в .NET Framework, є однією з сильних сторін платформи. Оскільки весь код збирається в байтовий код CIL, .NET насправді не дуже важливо, на якій мові його вписувати; можна обирати мову, виходячи з власних уподобань, різних сильних і слабких сторін кожної мови або різних аспектів .NET Framework, до якої надається доступ до кожної мови (тут є деякі варіанти). Як зазначалося, більша частина стандартної бібліотеки була написана на C #, але це не заважає отримувати доступ до цих класів з коду, написаного на інших мовах CLI. Дійсно, компоненти, написані на різних мовах CLI, можуть вільно працювати в додатку .NET.

З всесвітньою популяризацією даної платформи в межах операційної системи Windows постало питання створення платформи і на інші операційні системи.

.NET Core – це модульна платформа, яка захоплює широкий спект технологій на основі яких можна розробляти і створювати продукти від дата-центрів до сенсорних пристроїв. Дана платформа підтримувана Microsoft, на Linux, Windows, Mac OSX і є доступною з відкритим вихідним кодом.

Станом на сьогодні для розробки клієнтської частини в межах платформи .NET в сучасному IT світі виділяють наступні технології WinForm, WPF.

Windows Forms – технологія для розробки графічних віконних додатків користувача є частиною Microsoft .NET Framework. За допомогою інтерфейсу створеного на основі технології Windows Forms спрощується доступ до графічник елементів Microsoft Windows за допомогою реалізації обгортки до Win32 API(інтерфейс програмування додатків)[14] в керованому коді. До тогож даний керований код є незалежним відносно мови програмування і може використовувати дану технологію як і на VB.Net, J # так і на C++, C# та ін.

Аналізуючи дану технологію можна дійти до висновку, що Windows Forms сприймається як альтернатива застрілої бібліотеки MFC, але дана технологія не пропонує ніякої парадигми, яка б могла порівнюватись з MVC. Тому для вирішення даної проблеми були створено сторонні бібліотеки. Однією із найбільш використовуваних можна вважати User Interface Process Application Block, яка була реалізованою компанією Microsoft.

Технологія WPF(Windows Presentation Foundation) – технологія для розробки десктопних графічних додатків користувача і є частиною платформи .NET.

Додатки, реалізовані на технології WPF засновані на основі DirectX. Це велика перевага над Windows Forms, адже там за відображення контролів(елементів управління і графіки) відповідали User32 і GDI+. Це говорить про те, що відображення елементів упраління, 3D графіки буде виконуватись за допомогою графічного процесора на відеокарті. Даний підхід дозволяє скористатись прискоренням апаратної графіки.

Однією з важливих особливостей є використання мови декларативною розмітки інтерфейсу XAML, заснованого на XML: ви можете створювати насичений графічний інтерфейс, використовуючи або декларативне оголошення інтерфейсу, або код на керованих мовах C # і VB.NET, або поєднувати і те, і інше.

Порівнюючи технорлогії WPF і WinForms можна дійти по висновку, що WPF має ряд переваг стосовно іншої технології:

- можливість розробляти додатки під безліч ОС сімейств Windows;
- прискорення апаратне графіки. Від 2D і до 3D графіки контроли додатку конвертуються в об’єкти, які в свою чергу є зрозумілими для Direct3D, а далі завдання візуалізації лягає на процесор відеокарти. Даний дає більшу продуктивність роботи графіки і робить її більш плавною;
- додається можливість для створення різнотипних додатків: двовимірна і тровимірна графіка, мультимедіа, великий набір вже готових елементів управління, а таож можливість створювати власні(кастомні) елементи;
- можливсть взаємодії із Windows Forms. Можливість управляти і додавати елменти із Windows Forms;

- можливість створення тривимірної графіки, прив’язки даних(Binding), можливість створювати базові стилі і шаблони;
- незалежність від розширення екрана. Додатки WPF, легко розробляти відносно різних розширень в порівнянні із Windows Forms;
- можливість створювати інтерфейс за допомогою спеціалізованої мови розмітки XANL, яка заснована на основі xml, має можливість створювати об’єкти, які конвертуються в C#/ VB.NET.

У той же час WPF має певні обмеження. Попри те, що WPF підтримує 2D і 3D графіку і може реалізовувати додатки, які міститимуть в собі мультимедіа, 2D і 3D анімацію, дана технологія не є оптимальною для створення ігор. Для реалізації даного типу завдань використовують спеціальні фреймворки Uniti чи Monogame.

Слід ще врахувати факт того, що додатки створені на WPF споживають більшу кількість пам’яті в порівнянні з технологією WindowsForm, але це можна компенсувати тим, що WPF представляє більшу кількість графічних елментів управління і надає більші можливості по створенню графічного інтерфейсу.

В процесі порівняння продуктивності роботи WPF і WinForms[15] можна виділити явну перевагу технології WPF над своїм аналогом рис 2.9.

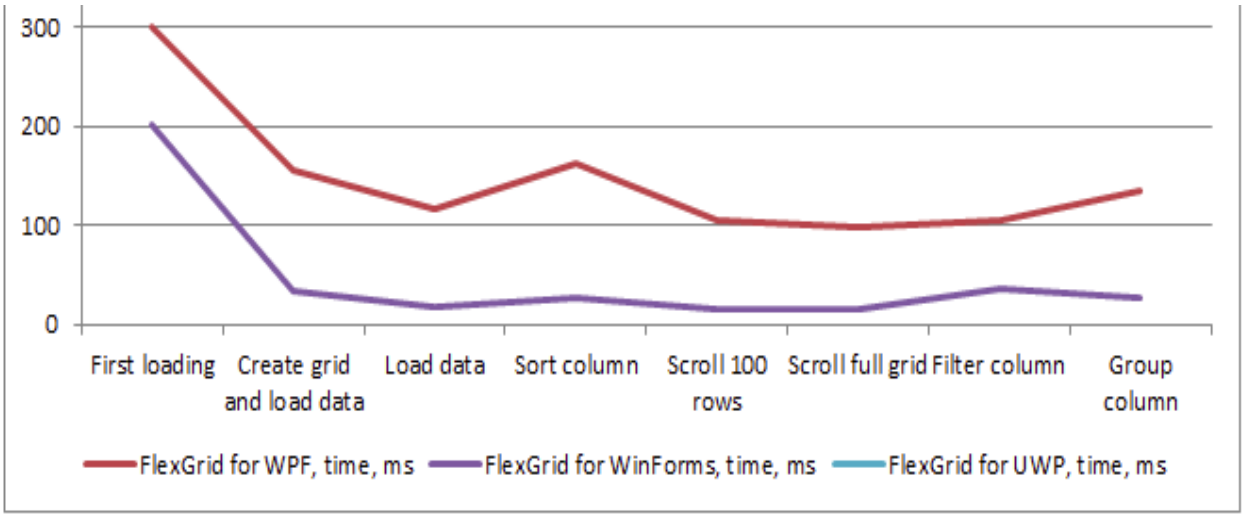


Рисунок 2.9 – Графік порівняння швидкодії технологій WPF і WinForms.

В результаті огляду і аналізу описаних вище технологій в межах платформи .NET для реалізації клієнтської частини додатку було обрано технологію WPF.

Для реалізації серверної частини додатку в межах архітектурного рішення, на даний момент часу виділяють наступний набір технологій Spring (Java), ASP.NET Core (C#) , GGI (C++). Розглянемо дані технології більш детально.

Технологія Java Server Pages (JSP) є технологогією за допомогою якої створюють веб застосування, які базуються на основі J2EE і використовують веб інтерфейс. В архітектурному плані JSP можна розглядати як абстракцію сервлетів Java на високому рівні. JSP перекладаються в сервлети під час виконання, тому JSP є сервлетом; кожен сервлет JSP кешується та використовується повторно, поки не буде змінено оригінальний JSP. [16]

JSP може використовуватися незалежно або як компонент перегляду в дизайні сервера на MVC[17], як правило, з JavaBeans як модель та сервлети Java (або рамки, такі як Apache Struts) як контролер.

JSP дозволяє управляти коду Java над деяким статичним вмістом веб-розмітки, таким як HTML, із отриманою сторінкою. Скомпільовані сторінки, а також будь-які залежні бібліотеки Java містять байт-код Java, а не машинний код. Як і будь-яка інша програма Java, вони повинні бути виконані в рамках віртуальної машини Java (JVM), яка взаємодіє з операційною системою сервера-хоста, щоб забезпечити абстрактне, нейтральне для платформи середовище.

JSP зазвичай використовуються для доставки HTML і XML документів, але за допомогою OutputStream[18] вони також можуть доставляти інші типи даних.

Веб-контейнер створює неявні об'єкти JSP, такі як запит, відповідь, сесія, додаток, конфігурація, сторінка, pageContext, та інші. Також даний фреймворк слід віднести до ряду архітектурно коректних, адже він побудований за допоогою основних принципів S.O.L.I.D. Даний підхід надає можливість гнучкості коду і робить можливим маштабування додатку. Порівняльна характеристика фреймворків наведена на рис. 2.10.

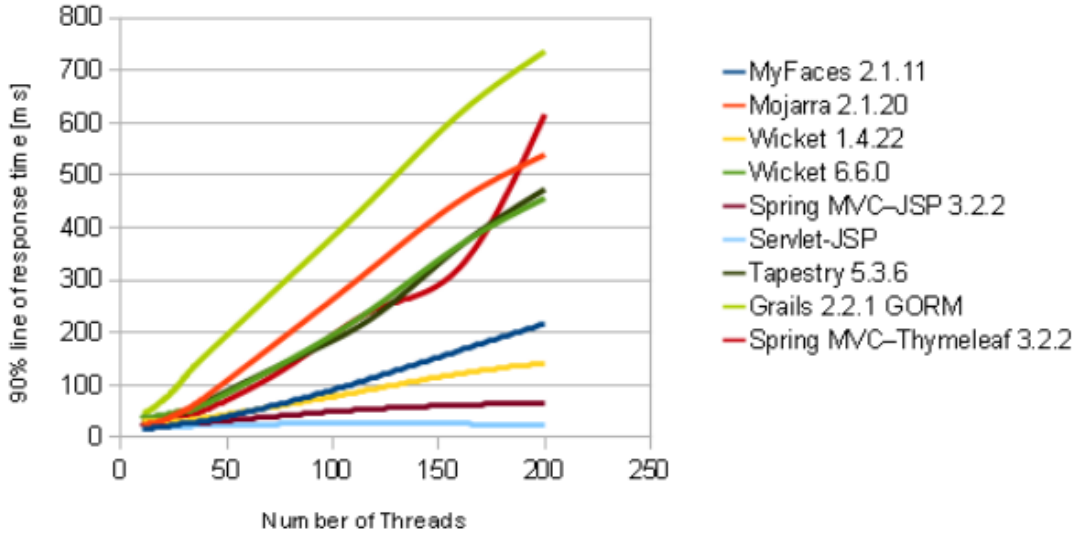


Рисунок 2.10 – Порівняння фреймворків.

Персонально широко використовувана технологія стала CGI (загальний інтерфейс шлюзу), яка спеціально застосовує для створення динамічних веб-сторінок і працює для забезпечення зв'язку між клієнтами (веб-браузером) і веб-серверами. Дана технологія являє собою набір правил, дотримуючись яких, програма здатна виконуватись на різних серверах операційних системах. Відповідно до технології CGI, HTTP запит, який містить посилання на динамічну сторінку, вступаючи на веб-сервер, генерує новий процес і запускає потрібну прикладну програму. Технологія CGI дозволяє використовувати будь-яку мову програмування, здатну працювати з пристроями введення / виводу. Також для розробки веб-додатків можна використовувати скрипти CGI, наприклад Python, Perl, Tcl і т. д. Якщо в програмі CGI міститься скрипт, то при його виконанні виводиться script engine (інтерпретатор скриптів), який передає дані HTTP запиту і імені файлу, що містить запитуваний скрипт. Після виконання даних скрипту програмної клієнтури повертається сформована HTML-сторінка.

Не дивлячись на те, що технологія CGI дозволяє досить просто створити інформацію про динамічну технологію у веб-мережах, вона має значні недоліки. Одним із головних недоліків є виробництво. Причиною низької продутивності є сам процес обробки HTTP запиту: для кожної обробки подібного запиту Веб-сервер генерує новий процес, який закінчує свою роботу тільки після завершення програм,

що є достатньо трудомістким і при наявності великої кількості таких процесів, розпочинаєиться конкуренція за ресурси оперативної пам’яті.

Однією із найвідоміших технологій для розробки веб додатків є ASP.NET Core. Ця нова платформа з відкрим кодом являється кросплатформеною. Програми ASP.NET Core можуть працювати в .NET Core або на повній .NET Framework. Він був розроблений, щоб забезпечити оптимізовану розробку для додатків, які розгорнуті на сервері або запущені локально. Він складається з модульних компонентів з мінімальними витратами, тому зберігається гнучкість при конструюванні рішень. Можна розробляти та запускати додатки ASP.NET Core у Windows, Mac та Linux. ASP.NET Core є відкритим кодом у GitHub.

Порівнюючи технології GGI та ASP.NET Core [19] можна дійти до висновку, що можливості по реалізації значно більші в межах задач системного програмува, програмування контролерів та мікропроцесорної техніки. Відповідно до поставленої задачі, її реалізація на c++ витратить набагато більше часу на фоні реалізації на високорівневій мові на кшталт JAVA .Net. Порівняння швидкості опрацювання одного і того самого запиту сервером реалізованим на c++ та ASP.Net зображено на рис 2.11

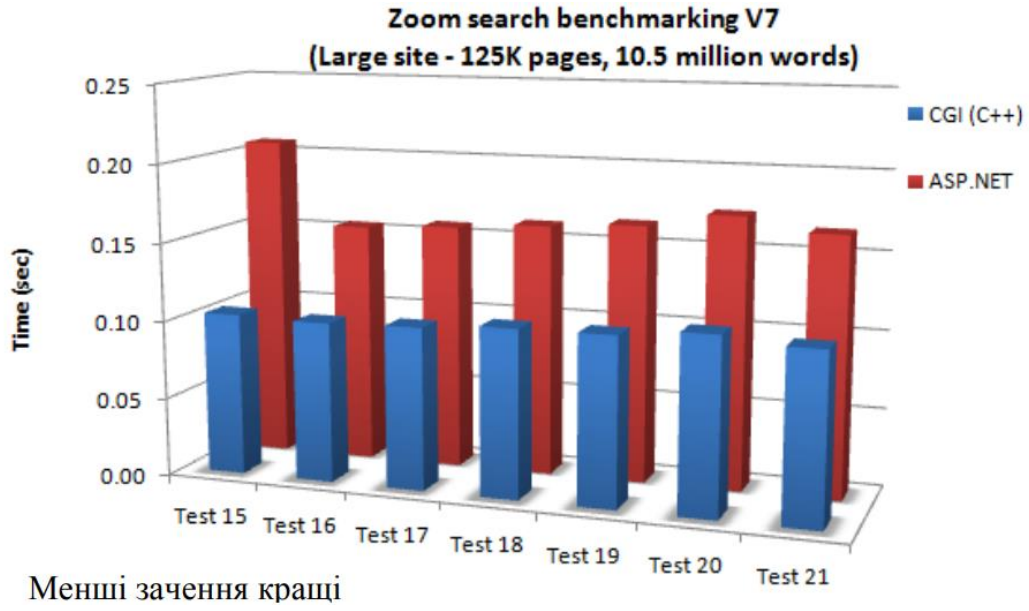
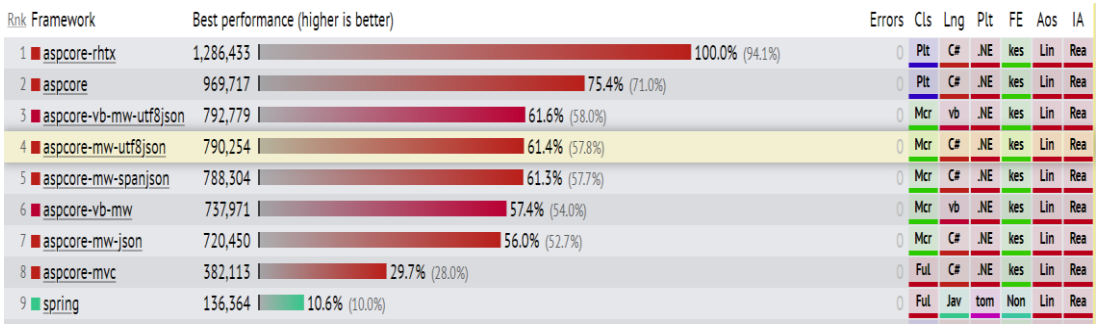


Рис 2.11 - Порівняння швидкодії реалізацій серверних рішень CGI та ASP.NET CORE

В межах огляду було зроблено порівняння техгології, які написані на базі Spring і ASP.NET Core [20] по критеріям швидкості перетворення структури даних у послідовність бітів (серіалізація) результати якого можна подивитись на рис. 2.12



Rnk	Framework	Best performance (higher is better)	Errors	Cls	Lng	Plt	FE	Aos	IA
1	aspcore-rhtx	1,286,433 100.0% (94.1%)	0	Plt	C#	.NE	kes	Lin	Rea
2	aspcore	969,717 75.4% (71.0%)	0	Plt	C#	.NE	kes	Lin	Rea
3	aspcore-vb-mw-utf8json	792,779 61.6% (58.0%)	0	Mcr	vb	.NE	kes	Lin	Rea
4	aspcore-mw-utf8json	790,254 61.4% (57.8%)	0	Mcr	C#	.NE	kes	Lin	Rea
5	aspcore-mw-spanjson	788,304 61.3% (57.7%)	0	Mcr	C#	.NE	kes	Lin	Rea
6	aspcore-vb-mw	737,971 57.4% (54.0%)	0	Mcr	vb	.NE	kes	Lin	Rea
7	aspcore-mw-json	720,450 56.0% (52.7%)	0	Mcr	C#	.NE	kes	Lin	Rea
8	aspcore-mvc	382,113 29.7% (28.0%)	0	Ful	C#	.NE	kes	Lin	Rea
9	spring	136,364 10.6% (10.0%)	0	Ful	Jav	tom	Non	Lin	Rea

Рис. 2.12 Порівняння швидкодії серіалізації структур даних на базі ASP.NET Core та системи на базі Spring

Аналізуючи результати тестів можна прийти до висновку, що для реалізації поставленої задачі в межах спроектованої архітектури для створення серверної частини системи найкраще підходить технологію ASP.NET Core.

3 ПРОГРАМНЕ РІШЕННЯ

3.1. Архітектура програмного рішення

Для рішення поставленого завдання було обрано клієнт – серверну архітектуру, серварна частина якої складається із SOA архітектури на основі мікросервісів. Серед загального шаблону обумовленого рішення можна виділити 3 основні коспоненти:

- клієнт;
- сервер;
- зовнішні API;

Розглянемо більш детально складові даної архітектури.

Клієнт (user agent) - це будь-який інструмент або пристрій, що діє від імені користувача. Це завдання переважно виконує веб-браузер; в деяких випадках

учасниками виступають програми, які використовуються інженерами і веб-розробниками для налагодження своїх додатків.

Браузер завжди є тією сутністю, яка створює запит. Сервер зазвичай цього не робить, хоча за багато років існування мережі були придумані способи, які можуть дозволити виконати запити з боку сервера.

Щоб відобразити веб сторінку, браузер відправляє початковий запит для отримання HTML-документа цієї сторінки. Після цього браузер вивчає цей документ, і запитує додаткові файли, необхідні для отображенія змісту веб-сторінки (виконувані скрипти, деталі компонування сторінки - CSS таблиці стилів, додаткові ресурси у вигляді зображень і відео-файлів), які безпосередньо є частиною вихідного документа, але розташовані в інших місцях мережі. Далі браузер з'єднує всі ці ресурси для відображення їх користувачеві у вигляді єдиного документа - веб-сторінки. Скрипти, що виконуються самим браузером, можуть отримувати по мережі додаткові ресурси на наступних етапах обробки веб-сторінки, і браузер відповідним чином оновлює відображення цієї сторінки для користувача.

Веб-сторінка є гіпертекстових документом. Це означає, що деякі частини тексту, що відображається є посиланнями, які можуть бути активовані (зазвичай натисканням кнопки миші) з метою отримання і відповідно відображення нової веб-сторінки (перехід по посиланню). Це дозволяє користувачеві "переміщатися" по сторінках мережі (Internet). Браузер перетворює ці гіперпосилання в HTTP-запити і в подальшому отримані HTTP-відповіді відображає в зрозумілому для користувача вигляді.

В нашому варіанті реалізації клієнтом виступатиме віконний графічний додаток користувача, який складатиметься із розмітки XAML за допомогою якої буде реалізовуватись відображення додатку.

Міжкомпонентна взаємодія в більшості випадків реалізовується за допомогою протоколу протоколу HTTP або HTTPS[21].

На іншій стороні комунікаційного каналу розташований сервер, який обслуговує (англ. Serve) користувача, надаючи йому свої сервіси. З точки зору кінцевого користувача, сервер завжди є якоюсь однією віртуальною машиною,

повністю або частково генеруючою документ, хоча фактично він може бути групою серверів, між якими балансується навантаження, тобто перерозподіляються запити різних користувачів, або складним програмним забезпеченням, яке опитує інші комп'ютери (такі як кешуючі сервери, сервери баз даних, сервери додатків електронної комерції та інші).

Сервер не обов'язково розташований на одній машині, і навпаки - кілька серверів можуть бути розташовані (хоститись) на одній і тій же машині. Відповідно до версією HTTP / 1.1 і маючи Host заголовок, вони навіть можуть ділити одну і ту IP-адресу.

Між віконним графічним додатком користувача і сервером знаходяться велика кількість мережевих вузлів передаючих HTTP повідомлення. Через шаруватість структури, більшість з них оперують також на транспортному мережевому або фізичному рівнях моделі OSI[22], стаючи прозорим на HTTP шарі і потенційно знижуючи продуктивність. Ці операції на рівні додатків називаються проксі

Для реалізації міжеомпонентної взаємодії на основі протоколів HTTP\HTTPS використовується технологія REST API (Representational state transfer).

3.2 Діаграма компонентів

Загальну схему роботи продемонстрована на рисунку 3.1:

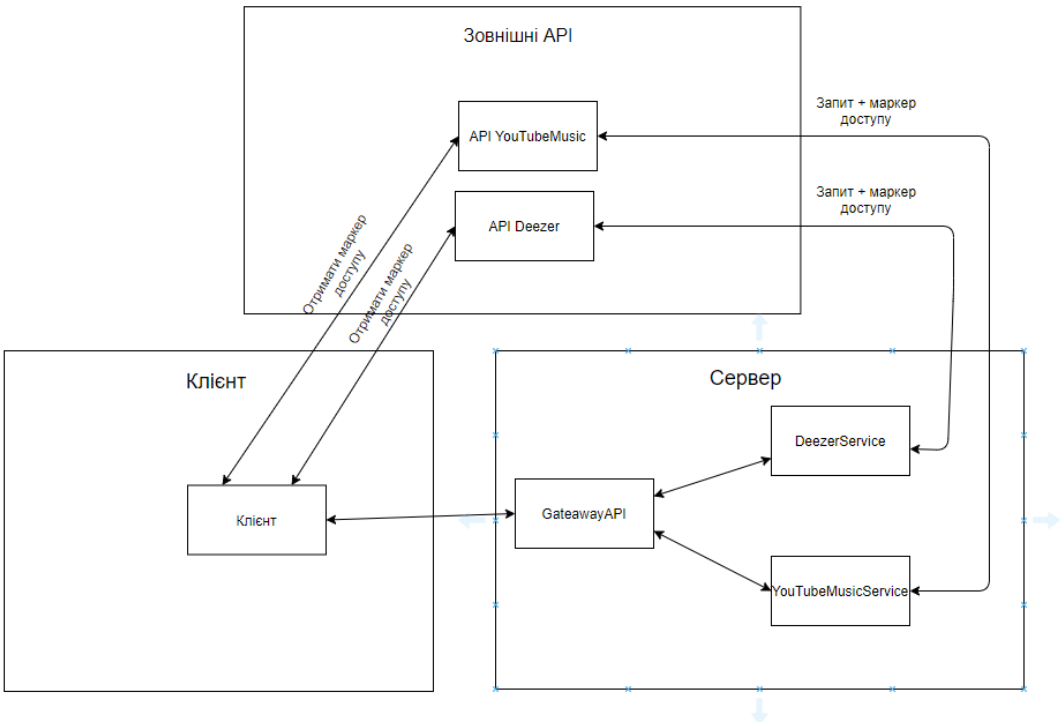


Рисунок 3.1 – Загальна схема роботи системи

3.3. Компонент серверу

Для реалізації серверної частини додатку було обрано мікросервісну архітектуру, яка складатиметься із сервіса шини даних, який відповідатиме за маршрутизацію пакетів між клієнтом і двома іншими мікросервісами. Для реалізації централізованого кервування музичними сервісами було обрано дві платформи: YouTube Music, Deezer. Враховуючи фактор загруженості і обробки великої кількості інформації в системах даного типу було прийнято рішення реалізувати незалежний мікросервіс для кожного музичного сервісу. В результаті ми отримали 3 мікросервіси в рамках обумовленого архітектурного рішення:

- ApiGeteway(Шина даних);
- YouTubeMusicService(Сервіс по обробці інформації для YouTube Music);
- DeezerService(Сервіс по обробці інформації для DeezerService);

Загальна схема архітектури відображена на рис 3.2

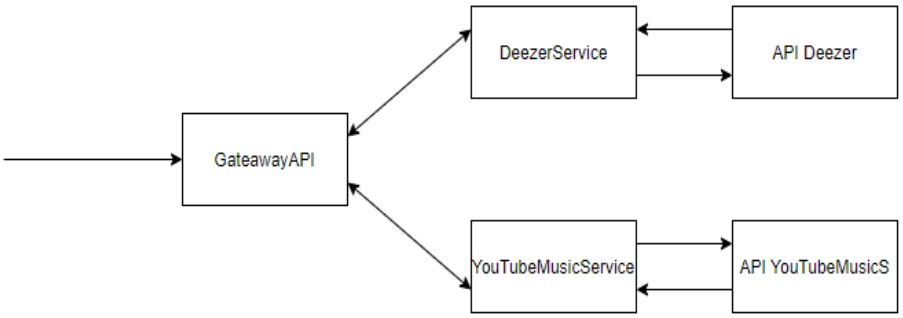


Рисунок 3.2 – Загальна схема сервеної частини додатку

Для реалізації шини даних в обумовленій архітектурі в рамках технології ASP.NET Core станом на сьогодні виділяють структуру Ocelot[23].

Ocelot - це структура з відкритим кодом, що використовується для побудови шлюзів API .NET Core, проект спрямований на людей, які використовують .NET / .NET Core для створення додатків, розроблених з мікросервісами або архітектурами SOA. Ocelot надає простий спосіб написання картографічного файлу (ocelot.json), який може бути використаний для маршрутизації вхідних запитів HTTP до відповідних сервісів.

Даний шлюз являється звичайним проксі-сервером, основна мета якого маршрутизація пакетів між мікросервісами. Реалізація даного завдання являється стандартизованим для більшості рішень і не містить проблем. Однак використання шлюзу API стає складнішим у реалізації, коли вони починають проводити агрегацію відповідей.

Агрегація відповідей - це техніка, що використовується для об'єднання відповідей з декількох сервісів нижче за потоком в один об'єкт. Шлюзи API досягаючи цього, приймаючи єдиний запит від клієнтів і видаючи кілька паралельних запитів до нижчих служб, як тільки всі сервіси нижче за течією відповідають, шлюзи API виконують об'єднання даних в єдиний об'єкт і подають їх клієнтам. Ця методика призводить до таких переваг:

- менше транзакцій HTTP між клієнтом і сервером. Оскільки відповідальність за надсилання запитів до декількох нижче розташованих служб перевантажується на шлюз API, зв’язок між клієнтом зводиться до меншої кількості транзакцій;
- клієнт повинен знати лише одну схему. Оскільки шлюзи API агрегують об'єкти відповідей, сторонам-клієнтам потрібно мати справу лише з єдиною формою даних порівняно зі схемою обробки багатьох постачальників;
- перехресна континентальна комунікація стає швидшою. Якщо клієнт, який звертається, знаходиться в іншій геолокації, коефіцієнт затримки, запроваджений міжконтинентальним зв’язком, стає меншим;

Для того аби реалізувати роботу з музичними сервісами YouTube Music і Deezer було оглянути і проаналізовано документації по роботі із відповідним зовнішнім API [24-25].

Для роботи із зовнішнім API для YouTube Music для платформи .NET компанія Google пропонує бібліотеку [26].

Клієнтські бібліотеки Google для .NET підтримують доступ до сервісів Google Cloud Platform таким чином, що значно скорочує об’єм вихідного коду, який потрібно реалізовувати. Бібліотеки надають абстракції API високого рівня, щоб їх було легше зрозуміти. Вони охоплюють ідіоми C #, добре працюють зі стандартною бібліотекою та краще інтегруються з базою кодів. Дана бібліотека спрощує роботу із сервісами YouTube.

Для роботи із платформою YouTube Music було реалізовано методи пошуку списка відтворення, отримання існуючих списків, треків, виконавців, а також створення власного списка.

Для роботи із сервісами Deezer було оглянуто і проаналізовано API документацію. Даний сервіс є схожим по роботі із YouTube Music:

- містить базовий URL для кожного методу API;
- має певні обмеження на кількість запитів в секунду(50 запитів на секунду);

- Структура відповіді доступна у форматах JSON, JSONP, XML, PHP[27];
- Усі запити і відпоіді повині бути у форматі UTF-8[28].

Для роботи із платформою Deezer було реалізовано методи за допогою яких можна отримати інформацію про авторизованого користувача, отримання списків відтворення і повної інформації про них, а також створення нових списків.

3.4 Рівень комунікації

Для взаємодії із серверною частиною додатку, а саме із мікросервісом шини даних, який приймає всі запити від клієнтів і перенаправляє їх до інших мікросервісів було створено URL – адресу до якої будуть спрямовувати запит і виглядатиме наступним чином: http://localhost:7000 для взаємодії по протоколу https адреса виглядатиме https://localhost:7001. Даний мікросервіс приймає всі запити клієнта і перенаправляє їх до двох інших сервісів: YouTubeMusicService, DeezerMusicService.

Структуру json для маршрутизації пакетів в рамках Ocelot обумовленої архітектури для роботи з мікросервісом DeezerService зображено на рис 3.3

```
{
  "DownstreamPathTemplate": "/api/{version}/{everything}",
  "DownstreamScheme": "http",
  "DownstreamHostAndPorts": [
    {
      "Host": "localhost",
      "Port": 5000
    }
  ],
  "UpstreamPathTemplate": "/api/{version}/{everything}",
  "UpstreamHttpMethod": [ "POST", "GET" ]
},
```

Рисунок 3.3 – Конфігурація маршрутизації на основі Ocelot

Розглянемо кожну складову більш детально:

- DownstreamPathTemplate, Scheme та DownstreamHostAndPorts роблять внутрішню URL-адресу мікросервісу, на яку буде спрямовано запит;
- Port - це внутрішній порт, який використовується сервісом. При використанні контейнерів порт вказаний у його докер файлі;

- Host - це ім'я служби, яке залежить від роздільної здатності назви служби, яка використовується;
- DownstreamHostAndPorts - це масив, що містить хост і порт будь-яких служб нижче, на які ви хочете пересилати запити. Зазвичай це поле буде містити лише один запис;
- UpstreamPathTemplate - це URL-адреса, яку використовуватиме Ocelot, щоб визначити, який DownstreamPathTemplate використовувати для даного запиту від клієнта. Нарешті, використовується UpstreamHttpMethod, щоб Ocelot міг розрізняти різні запити (GET, POST, PUT) до однієї і тієї ж URL-адреси.

Отже доступ до мікросервіса DeezerMusicService буде доступним за базовим посиланням http://localhost:5000 по протоколу http і за посиланням https://localhost:5001 по протоколу https. Базова структура запитів до мікросервісів виглядатиме наступним чином /api/{version}/{everything}, де version – назва контролера, який оброблятиме запит і everything назва метода, який виконуватиметься.

DeezerMusicService має 3 метода, які можна викликати:

- GetUser – даним метод приймає 1 параметр access_token який міститиме дані про маркер доступу. Даний метод повертає об’єкт DataResult<RootObjectUser> де RootObjectUser включає в себе всю інформацію про зареєстрованого користувача, а DataResult містить в собі цю інформацію при успішності виконання або при виникненні помилки містить її опис;
- GetPlayList – вхідний параметри метода access_token, який містить інформацію про маркер доступу. Метод повертає значення DataResult< RootObjectPlayList >, де RootObjectPlayList – список відтворення, який містить всю інформацію про треки, виконавців і тд;
- CreatePlayList – вхідні параметри access_token(маркер доступу), title(назва списку відтворення). Метод повертає об’єкт

DataResult<CreatePlayList> де CreatePlayList об’кт, який містить id стореного списку відтворення при успішній операції.

Доступу мікросервіса DeezerMusicService до зовнішнього API Deezer відбувається за базовим посиланням https://api.deezer.com/. Для того аби отримати інформацію про користувача і його списків відтворення було використано GET запити, які були доступні за допомогою наступних URL – посилань сервіса Deezer:

- https://api.deezer.com/user/{user_id} – посилання для отримання інформації про користувач;
- https://api.deezer.com/{user_id}/playlists – посилання для отримання інформації про списки відтворень;
- Кожен із списків відтворень містить посилання за яким можна отримати всі тректи даного списка.

Для створення списку відтворення було використано Post запит, який доступний за URL – посиланням https://api.deezer.com user/{user_id}/playlists в тілі даного запита передається поле title в якому міститиметься назва створеного списку відтворення.

Бащове URL для сервіса YouTubeMusicService по протоколу http: https://localhost:8004 по протоколу https і https://localhost:8006. Базова структура запитів до мікросервісів виглядатиме наступним чином /api/{version}/{everything}, де version – назва контролера, який оброблятиме запит і everything назва метода, який виконуватиметься. Даний мікросервіс має один контроллер YouTubeMusic і приймає 2 Post запити:

- GetPlayList – метод приймає об’єкт GoogleToken, який мість маркер доступу, id даного маркера, а також час існування токена. Метод всю інформацію про списки відтворення;
- CreatePlayList – метод приймає об’єкт PlayListDataToCreate, який містить GoogleToken, список треків, а також назву списку.

Для взаємодії із зовнішнім API сервіса YouTubeMusic було використано бібліотеку Google.Apis.YouTube.

3.5. Опис кліента

Для реалізації клієнтської частини додатку було обрано технологію WPF. В межах технології на даних момент часу для розділення відображення і бізнес логіки виділяють шаблон MVVM.

Даний шаблон був представлений Джоном Госсманом (John Gossman) в 2005 році як модифікація шаблону Presentation Model і був спочатку націлений на розробку додатків в WPF. І хоча зараз цей шаблон вийшов за межі WPF і застосовується в самих різних технологіях, в тому числі при розробці під Android, iOS, проте WPF є досить показовою технологією, яка розкриває можливості даного шаблону.

Даний шаблон складається із трьох компонентів:

- модель(Model);
- модель-відображенн(ViewModel);
- відображення(Вид).

Модель описує використовувані в додатку дані. Моделі можуть містити логіку, безпосередньо пов'язану цими даними, наприклад, логіку валідації властивостей моделі. У той же час модель не повинна містити ніякої логіки, пов'язаної з відображенням даних і взаємодією з візуальними елементами управління.

Нерідко модель реалізує інтерфейси INotifyPropertyChanged або INotifyCollectionChanged, які дозволяють повідомляти систему про зміни властивостей моделі. Завдяки цьому полегшується прив'язка до подання, хоча знову ж пряму взаємодію між моделлю і представленням відсутня.

View або подання визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Стосовно до WPF уявлення - це код в xaml, який визначає інтерфейс у вигляді кнопок, текстових полів та інших візуальних елементів.

Хоча вікно (клас Window) в WPF може містити як інтерфейс в xaml, так і прив'язаний до нього код C #, проте в ідеалі код C # не повинен містити якийсь логіки, крім хіба що конструктора, який викликає метод InitializeComponent і

виконує початкову ініціалізацію вікна . Вся ж основна логіка додатки виноситься в компонент ViewModel.

Однак іноді в файлі пов'язаного коду все може знаходитися певна логіка, яку важко реалізувати в рамках паттерна MVVM у ViewModel.

ViewModel або модель уявлення пов'язує модель і уявлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу INotifyPropertyChanged автоматично йде зміна відображуваних даних в поданні, хоча безпосередньо модель і уявлення не пов'язані.

ViewModel також містить логіку по отриманню даних з моделі, які потім передаються в уявлення. І також VewModel визначає логіку по оновленню даних в моделі.

Оскільки елементи уявлення, тобто візуальні компоненти типу кнопок, не використовують події, то уявлення взаємодіє з ViewModel за допомогою команд.

Наприклад, користувач хоче зберегти введені в текстове поле дані. Він натискає на кнопку і тим самим відправляє команду під ViewModel. А ViewModel вже отримує передані дані і відповідно до них оновлює модель.

Отже в результаті аналізу і огляду даного шаблона можна зробити висновок, що застосування шаблону MVVM є функціональний розподіл програми на три компонента, які простіше розробляти і тестувати, а також в подальшому модифікувати і підтримувати рис 3.4.

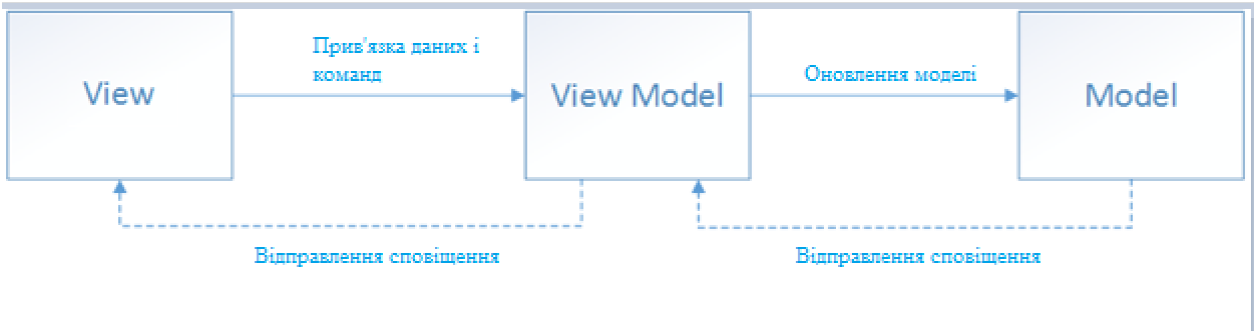


Рисунок 3.4 – Структура шаблона MVVM

Однією з найпопулярніших моделей асоціації View з View - Model є шаблон ViewModelLocator. Це особливий випадок шаблону пошуку локальних служб,

спеціально адаптованого до XAML та MVVM. Оновленні контроли підтримують цю схему з класом ViewModelLocatorBase.

Коли відображається View, він використовує ViewModelLocator, щоб знайти своє відповідне View - Model.

Додаток XAML створює екземпляр ViewModelLocator та додає його до свого словника ресурсів. Потім, це створює перший View. У View використовується ViewModelLocator, щоб знайти свою View-Model та встановити контекст даних. Коли настає час перейти до нового View, дочірній View проходить той самий процес рис 3.5.

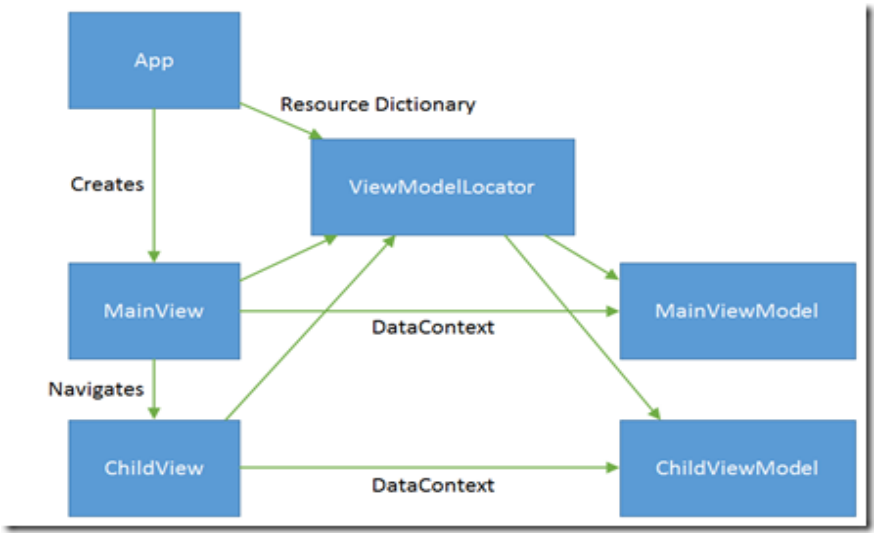


Рисунок 3.5 – Шаблон ViewModelLocator

Основна мета цього інструментарію - прискорити створення та розвиток додатків MVVM у Windows Universal, WPF, Silverlight, Xamarin.iOS, Xamarin.Android та Xamarin.Forms.

Інструментарій MVVM Light Toolkit допомагає відокремити View від Model, що створює додатки, які є більш чистими та легшими в обслуговуванні та розширенні. Він також створює тестовані програми та дозволяє мати значно тонший рівень користувальницького інтерфейсу (який складніше перевірити автоматично).

Цей інструментарій робить особливий акцент на спроможності створеного додатку (тобто здатності відкривати та редагувати інтерфейс користувача в Blend[29]).

3.3.1 Авторизація

Під час огляду музичних сервісів YouTube Music і Deezer було виявлено, що дані сервіси для авторизації користувачів використовують протокол OAuth2.0. Кожен із сервісів пропонує розробникам своє власне вікно авторизації рис 3.6, 3.7

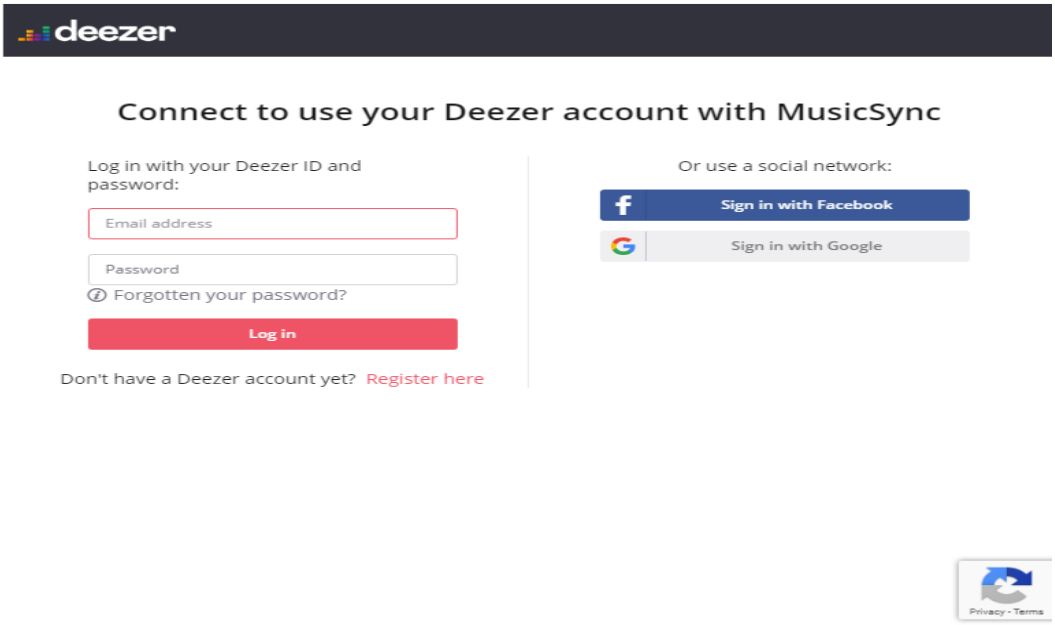
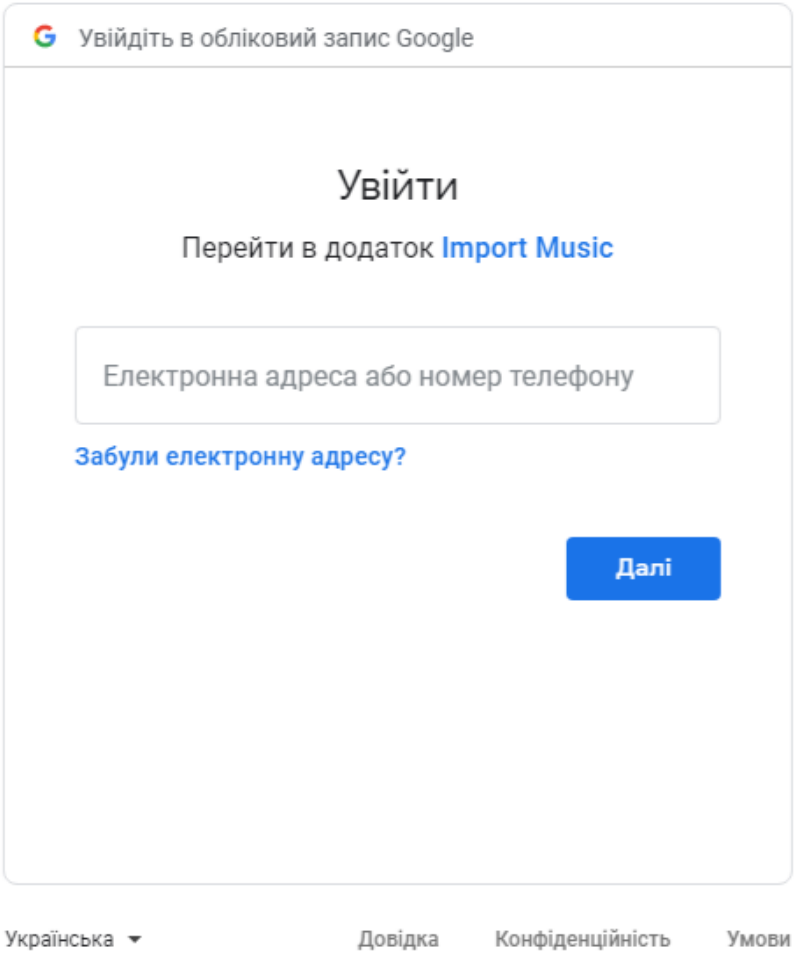


Рисунок 3.6 – Вікно авторизації для сервіса Deezer

Для інтеграції даних вікон авторизації до свого проекту необхідно зареєструвати його у ссервісах Google і Deezer. У сервісі Google додаток було зареєстровано під назвою Import Music див рис 3.5. Після вдалої реєстрації для додатка надається унікальний ідентифікатор клієнта client_id, унікальний ідентифікатор проекта project_id, секретний ключ для обміну повідомлень ClientSecret, URL - посилання на вікно авторизації та інше, посилання для перенаправлення після авторизації redirectURL та інше. Сервіс Deezer має схожий процес авторизації.

Після успішної реєстрації додатку в кожному із музичних сервісів, для реалізації авторизації в програмному продукті необхідно інтегрувати веб-браузер в додаток



.

Рисунок 3.7 – Вікно авторизації для сервісу YouTube Music

Для вирішення даного завдання на даний момент в рамках платформи .NET виділяють Chromium Embedded Framework.

Chromium Embedded Framework (CEF) - це проект з відкритими початковими кодами, створений у 2008 році як елемент управління Web browser, що працює на базі Chromium від Google.

На даний момент це досить потужний інструмент для розробки настільних додатків, зі списком рішень, що використовують цей елемент управління можна ознайоитися за посиланням [30]. Але досить сказати, що його використовують такі широко відомі продукти, як Evernote і Steam.

Даний фреймворк виконує наступні функції:

- CEF дозволяє створити свої обробники протоколів, таким чином, реалізувати свій «закритий» алгоритм шифрування;
- CEF дозволяє робити обгортку над нативними функціями в просторі об'єктів віртуальної машини Javascript. Ресурсомісткі операції по обробці великих масивів даних можна перекласти на більш швидкі мови програмування;
- CEF дозволяє обробляти події навігації, скачування файлів і так далі.
- загалом на основі даного фреймворка можна зробити свій власний браузер на зразок google chrome.

Даний фреймворк реалізований на мові програмування C++, але для C# розробників існує рішення у вигляді обгортки CefSgarp для .NET.

CefSharp - бібліотека-обгортка для chromiumembedded. Однак функціонал її дещо поступається за можливостями останньої. Для розробників доступні наступні функції:

- створення необмеженої кількості компонентів класу WebView;
- обробка подій по завантаженні сторінки, події навігації;
- власні обробники протоколів;
- впровадження js-коду під час виконання сторінки;
- створення глобальних [native code] об'єктів зі статичними методами.

Для інтеграції даної бібліотеки в наше рішення для реалізації авторизації музичних сервісів за допомогою NuGet пакетів було встановлено бібліотеку SefSharp.WPF. Для відображенн браузера в клієнтському вікні користувача було використано елемент управління ChromiumWebBrowser. Вікно авторизації для сервісів доступне за URL – посиланням, яке було отримано після реєстрації додатку в сервісах Google і Deezer. Для даного браузера був встановлений обробник подій Browser_FrameLoadStart, який оброблятиме всі відповіді браузера і після успішної авторизації клієнта отрмає код підтвердження(aprovalCode). Даний код являє собою код підтвердження успішності авторизації клієнта і за домогою якого можна отримати маркер доступу.

3.3.1. Services

Для реалізації клієнтської частини додатку виділено основні стани роботи додатку:

- Стан авторизації;
- Стан перегляду інформації про треки і списки відтворення;
- Стан парсингу даних з файла, URL-посилання та простого тексту;

Для кожного із станів реалізовується певний набір ViewModel:

- ViewModelLocator – головний клас в якому зберігаються всі інші;
- AuthorizationViewModel – клас для логіки авторизації;
- ConfigPlayListViewModel – клас для логіки конфігурації списку відтворення;
- ConfirmTrackViewModel – клас для підтвердження конфігурації списку відтворення;
- FileImportPageViewModel – клас для логіки парсинга файлів;
- MainViewModel – клас який містить у собі всю базову логіку додатку, а також всі інші класи по роботі із ViewModel;
- MenuViewModel – клас для роботи із меню користувача;
- PlainTextViewModel – клас для обробки інформації по введеному тексту корситувача;
- PlayListManagerNavigationViewModel – клас для міжвіконної новігації;
- UrlImportViewModel – клас для парсингу даних з URL – посилання;
- SelectMusicServiceViewModel – клас для вибору музичного сервісу в якому необхідно створити список відтворення.

Діаграму даних класів представлена на рис 3.6.

Рисунок 3.6 – Діаграма класів ViewModel

Далі розглянуто коротко деякі класи більш детальніше:

- Клас ViewModelLocator - даний клас містить в собі всю необхідну інформацію всіх вікон відображень і реєструє в собі всі використовувані в додатку сервіси за допомогою DI контейнерів[31];
- MainViewModel - клас, який містить в собі всі ViewModel, які використовуються в проекті, а також проводить ініціалізацію використовуваних сервісів;

- AuthorizationViewModel - даний клас містить інформацію про маркер доступу для сервіса YouTube Music, а також реалізовує зв'язок між сервісами передачі і отримування списків відтворення;
- AuthorizationViewModel – даний клас містить всю інформацію про відображення на головному вікні користувача: списки відтворень для YouTube Music і Deezer, доступні музичні сервіси, новігація між сторніками, а також параметри, які відповідають за відображення.

3.3.2 BL обробка запитів

Для взаємодії клієнтської частини із сервером було реалізовано базовий механізм на основі REST API і виділено базове URL посилання http://localhost:7000/api/. Були виділено базові сутності для передачі, а також сутності опису маркера доступу. Для реалізації передачі і отримання даних від сервера були реалізовані синхронні і асинхронні GET, POST методи в узагальненому абстрактному класі BaseApi. Від даного класу наслідуються класи YouTubeApi і Deezer Api, які включають конкретну реалізацію відправлення даних. Діаграма класів представлена на рис 3.7.

Для передачі даних між сервісами були виділені наступні сутності див рис 3.8

Для відображення інфомації про списки відтворення було виділено наступні сутності див рис 3.9

Дані сутності включають параметри, які необхідні для відображення клієнту, а також параметри, які використовуються для коректного графічного відображення.

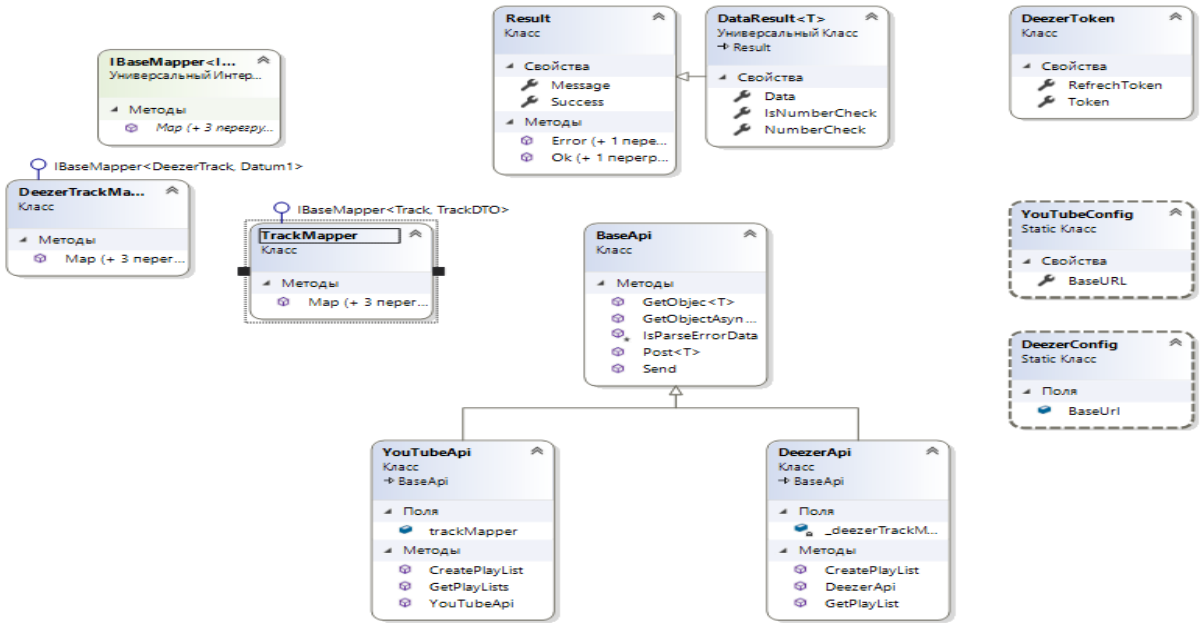


Рисунок 3.7 – Діагрма класів для комунікації із сервером.

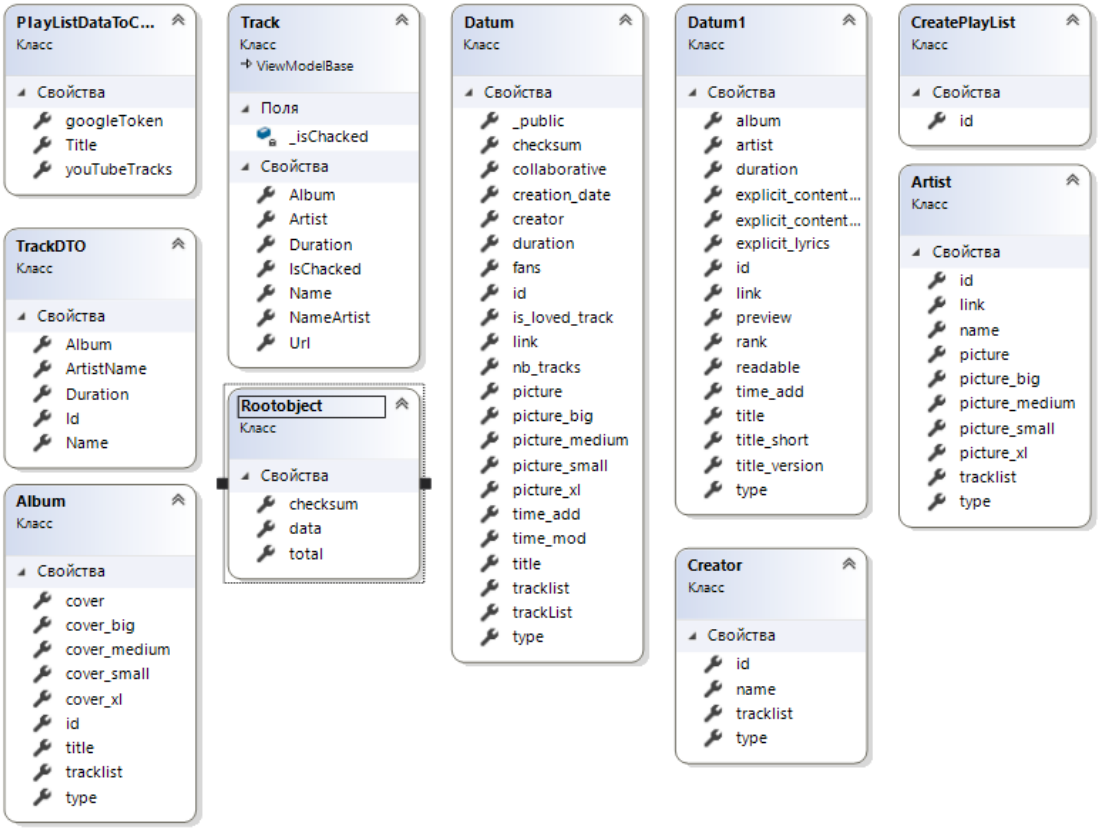


Рисунок 3.8 – Діаграма класів сутностей передачі даних

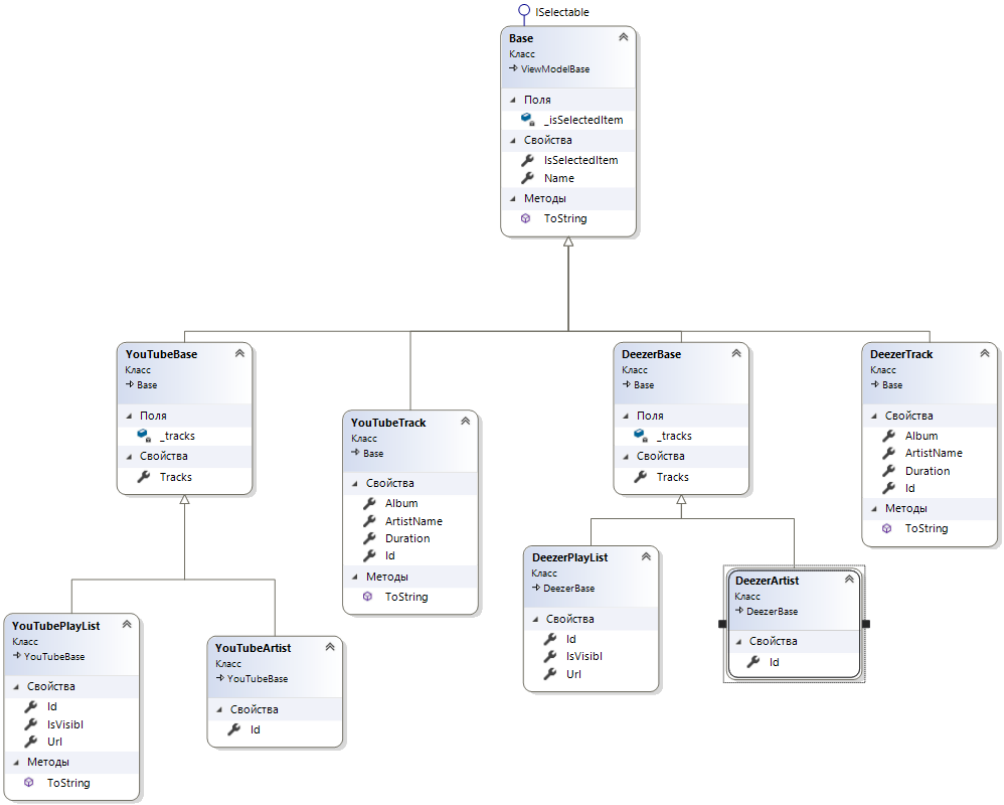


Рисунок 3.9 – Діаграма класів сутностей передачі даних

3.3.3 Представлення

Для візуалізації додатку було створено базове вікно MainWindow, а також 8 сторінок відображення де на головній сторінці MenuPage.xaml буде змінюватись відносно вибраного музичного сервіса 2 користувацькі елементи (User Control). Діаграма клісів для представлення рис 3.10.

Для реалізації парсингу різних типів файлів було використано шаблон абстрактна фабрика(Abstract Factory) [32].Даний шаблон створює загальний інтерфейс однотипних елементів, які є взаємопов’язаними між собою і реалізовують певні механізми із базового інтерфейсу.

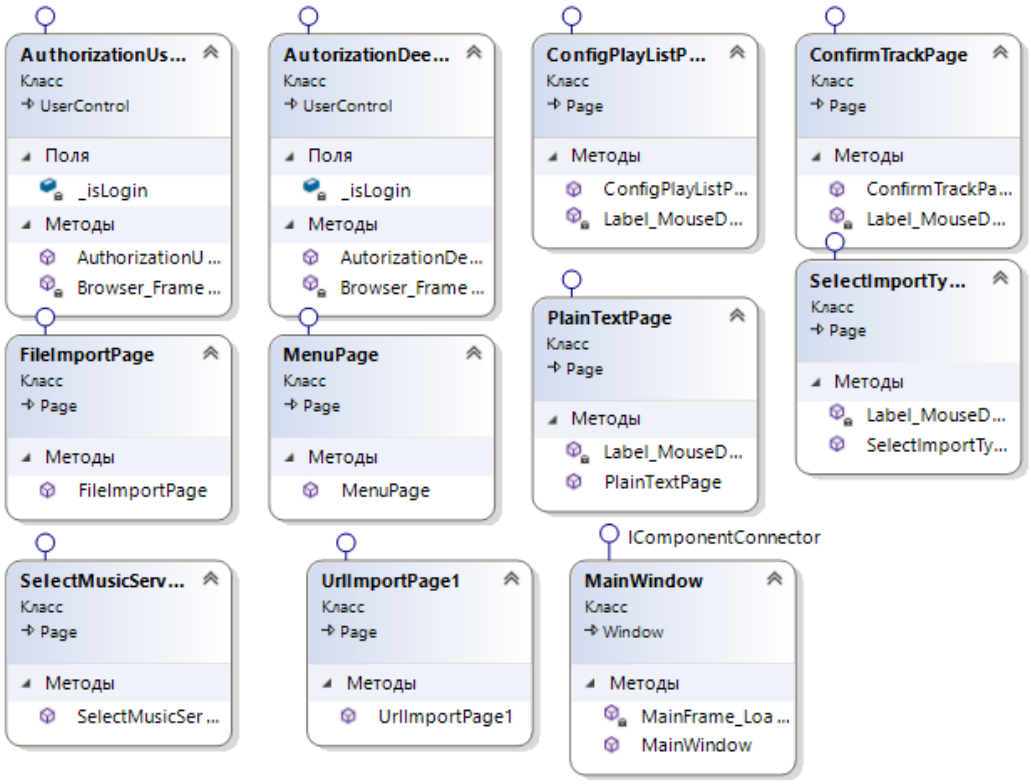


Рисунок 3.10 – Діаграма класів відображення

Використовується в наступних випадках:

- у випадках, коли ініціалізовані об’єкти є взаємопов’язаними між собою;
- коли система не залежить від способу створення нових об’єктів.

Загальний опис роботи шаблону продемонстрований на рис 3.11

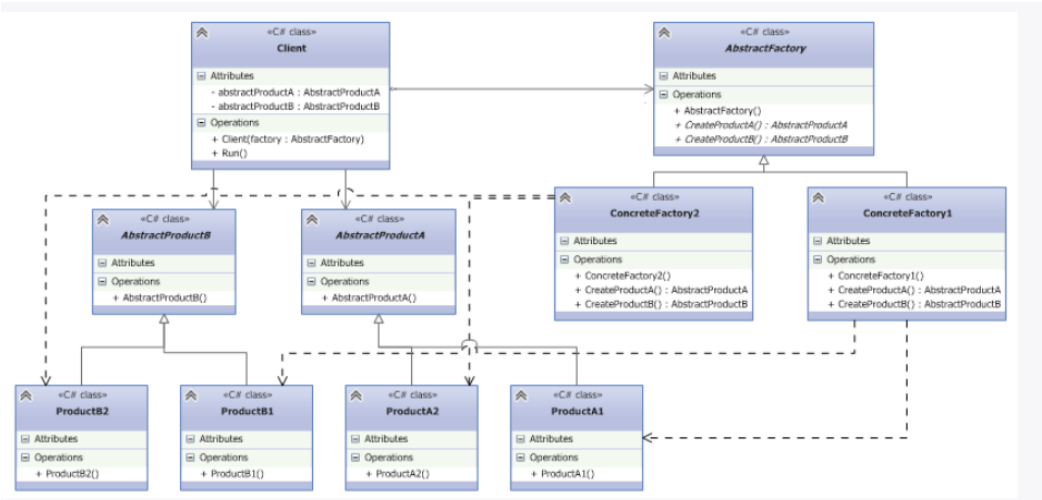


Рисунок 3.11 – Базовий опис шаблону «Абстрактна фабрика»

У нашому випадку ми маємо ряд однотипних класів, які використовують базовий інтерфейс парсинга файлів, але з модифікованою реалізацією. На рис. 3.12 зображено діаграму класів для парсинга різних типів файлі: json, m3u, txt, zpl.

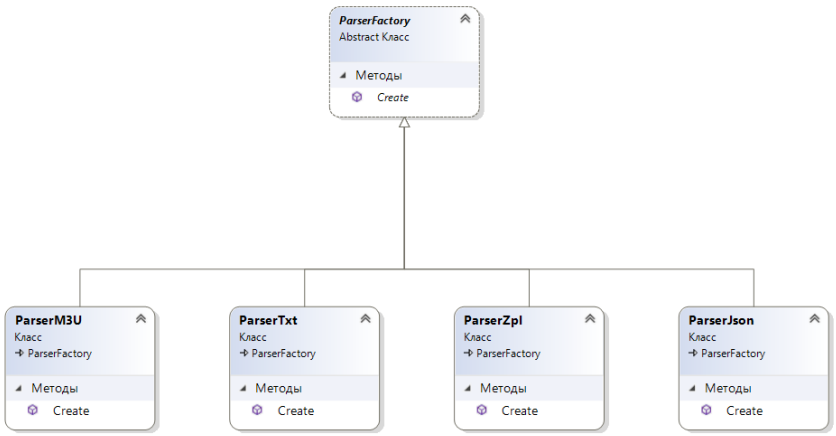


Рисунок 3.12 – Діаграма класів для парсинга файлів.

3.6 Виконана програма

Відображення всієї бізнес – логіки клієнтського додатку відображено на рис 3.13

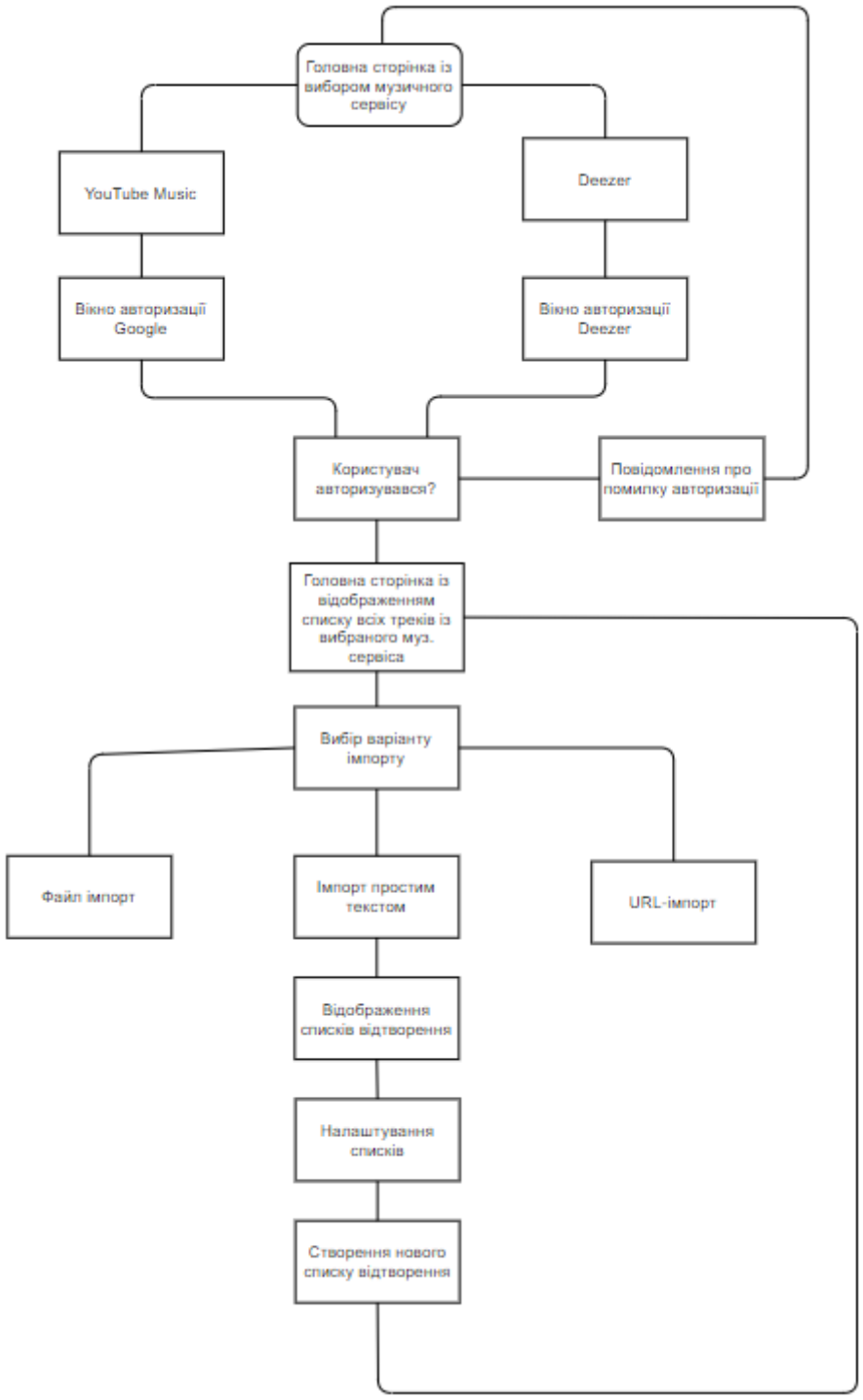


Рисунок 3.13 – Бізнес – логіка додатку

Під час запуску віконного графічного вікна користувача відкривається головне вікно програми із вибором авторизації для YouTube Music і для Deezer рис 3.14, 3.15.

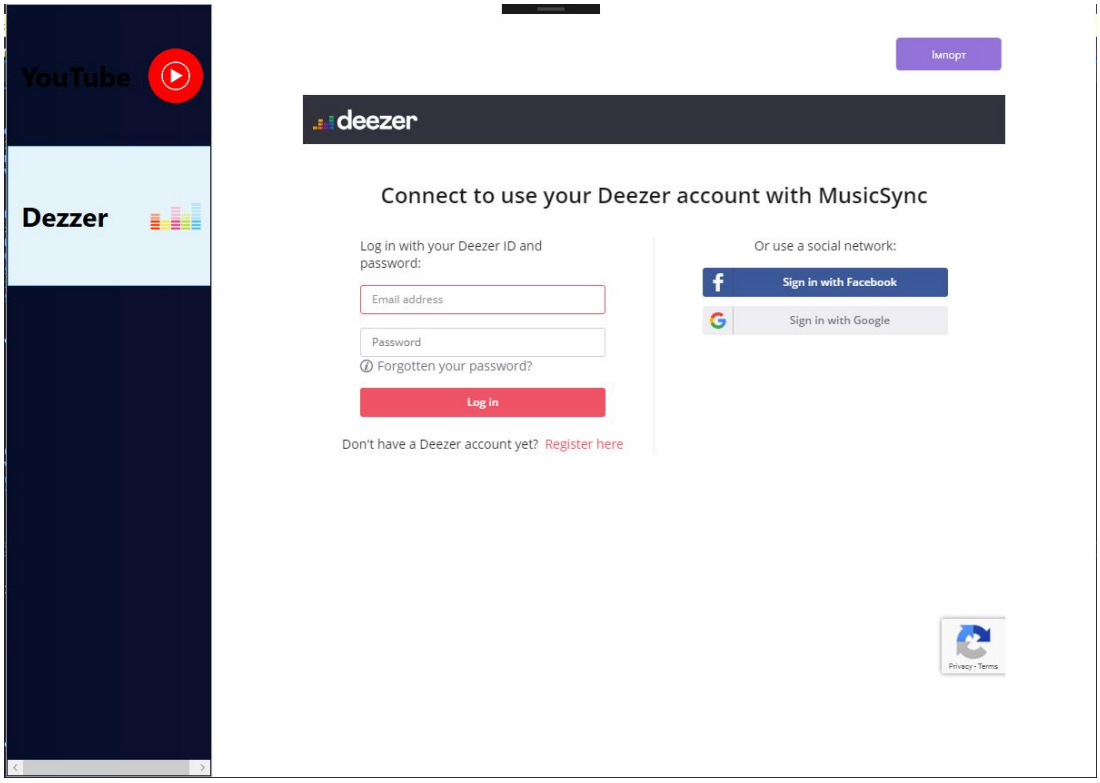


Рисунок 3.14 – Головне меню клієнта з авторизацією для Deezer

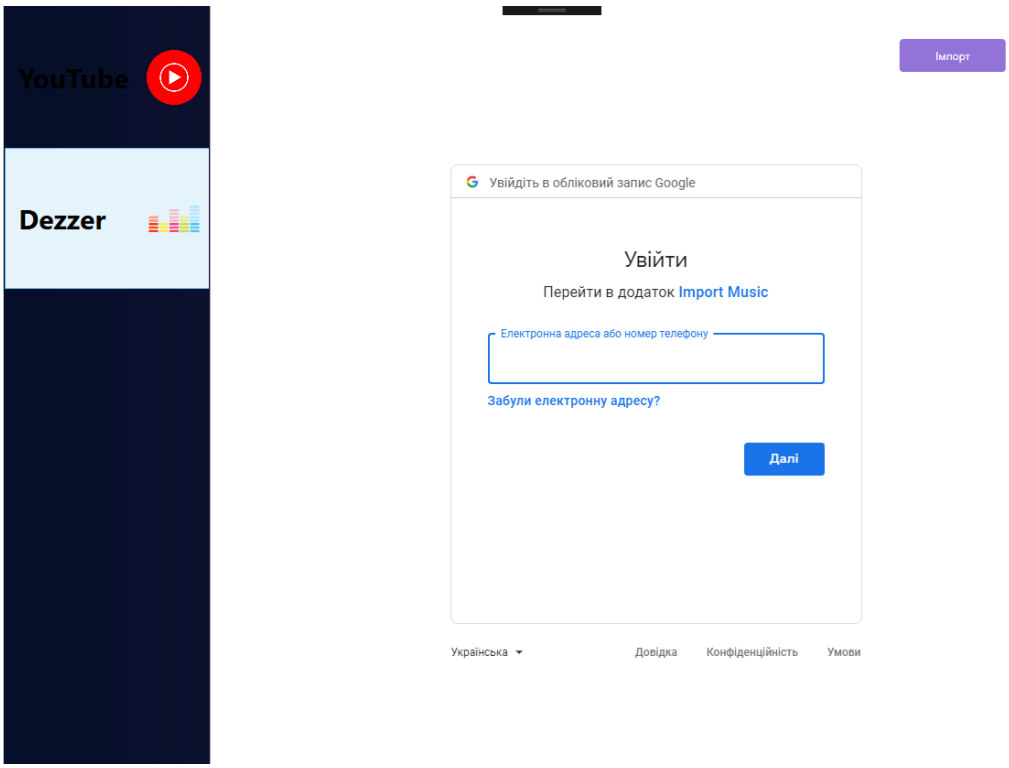


Рисунок 3.15 – Головне меню клієнта з авторизацією для YouTubeMusic

При успішній авторизації клієнт отримує маркер доступу, який далі відправляє на шину даних, яка в свою чергу перенаправляє запит до

YouTubeService, який обробляє його і повертає відповідь, шині даних, а вона відправляє клієнту дані про його списки відтворень і на стороні клієнта вони будуть відображатись наступним чином рис 3.16

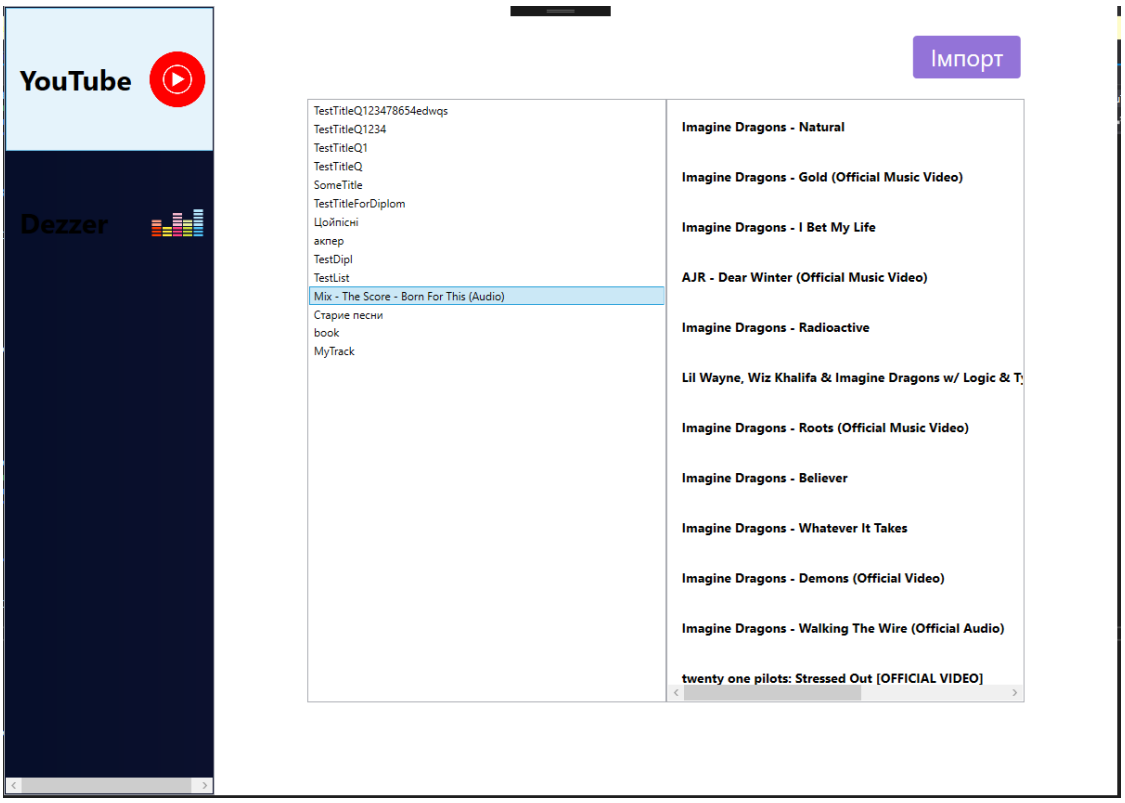


Рисунок 3.16 – головне меню із списками відтворень для YouTube Music

Натиснувши кнопку «Імпорт» для клієнта відображається вибір імпорту даних рис 3.17, 3.18, 3.19, 3.20

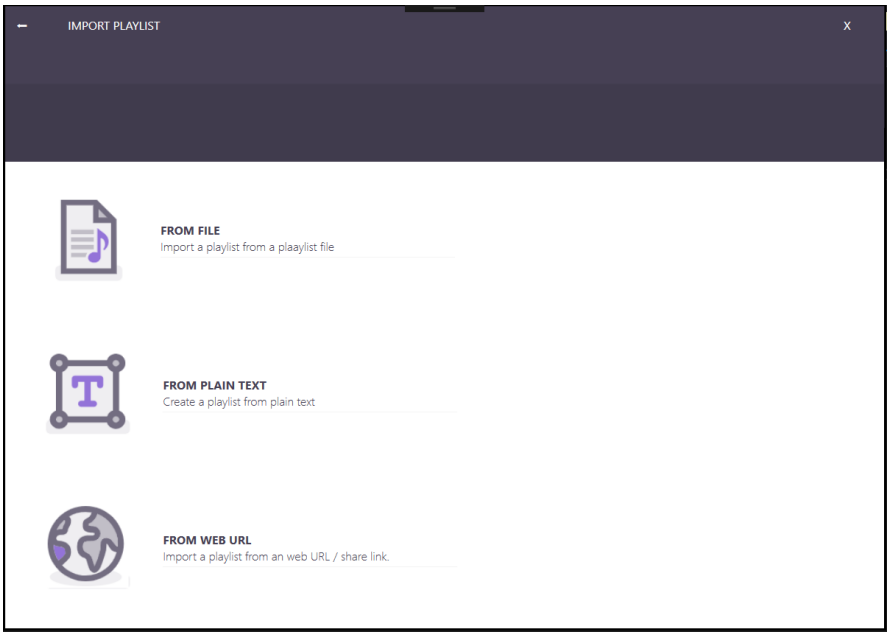


Рисунок 3.17 – Вікно вибору імпорту

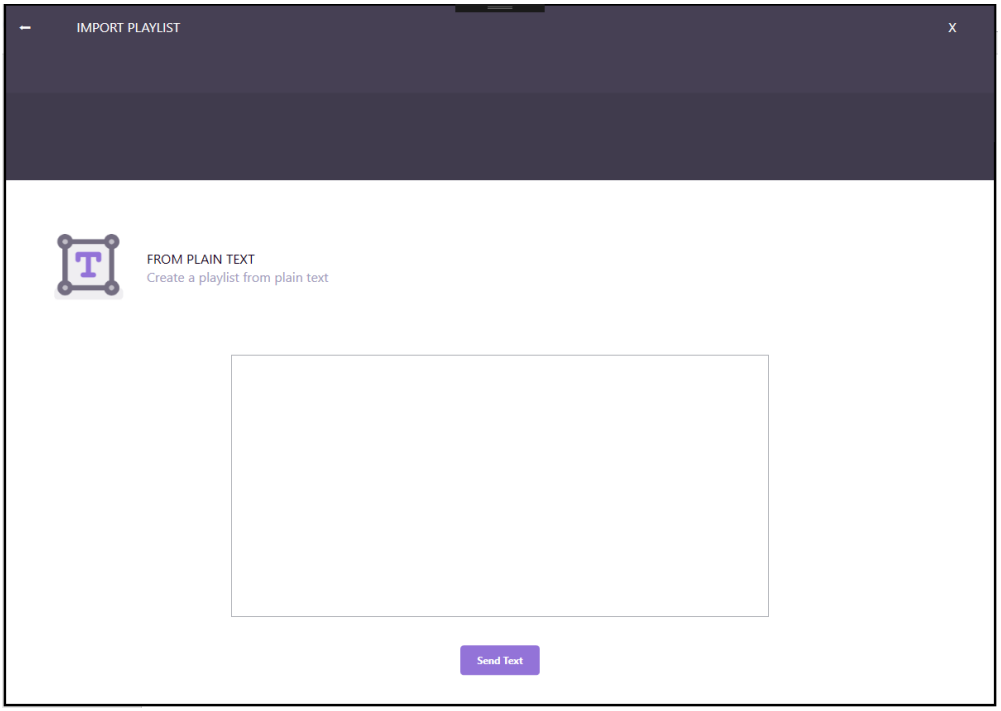


Рисунок 3.18 – Вікно вводу звичайного тексту

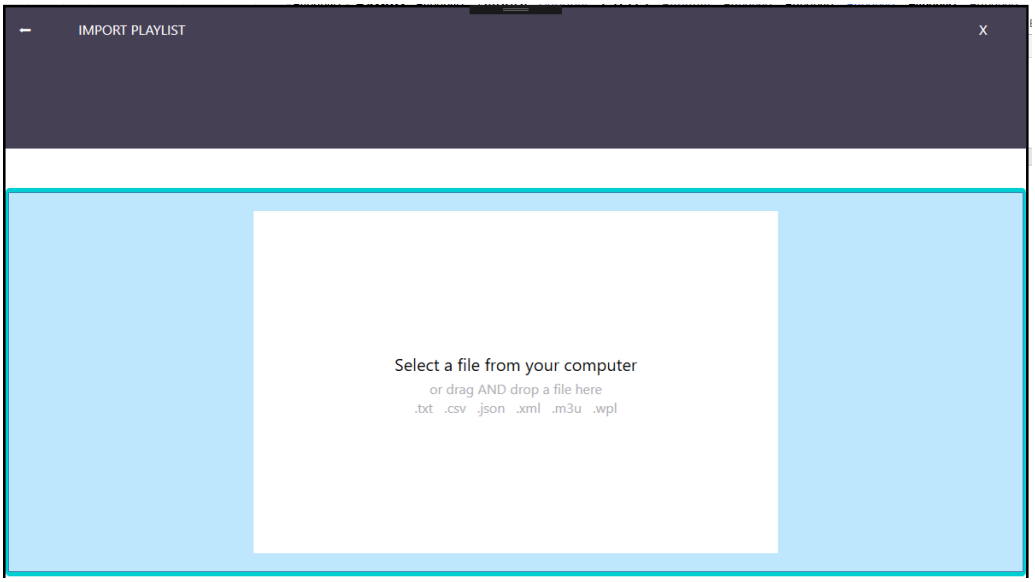


Рисунок 3.19 – Вікно вибору файла для імпорту

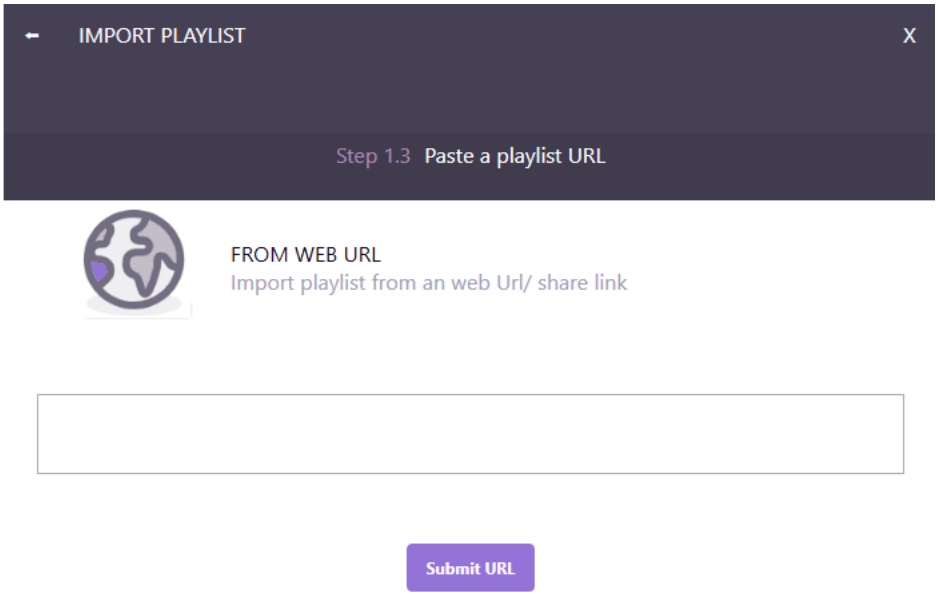


Рисунок 3.20 – Вікно введення URL – посилання для імпорту

Оберемо в якості імпорту даних – імпорт за допомогою введення ручного тексту і запишими 3 музикальні групи рис 3.21

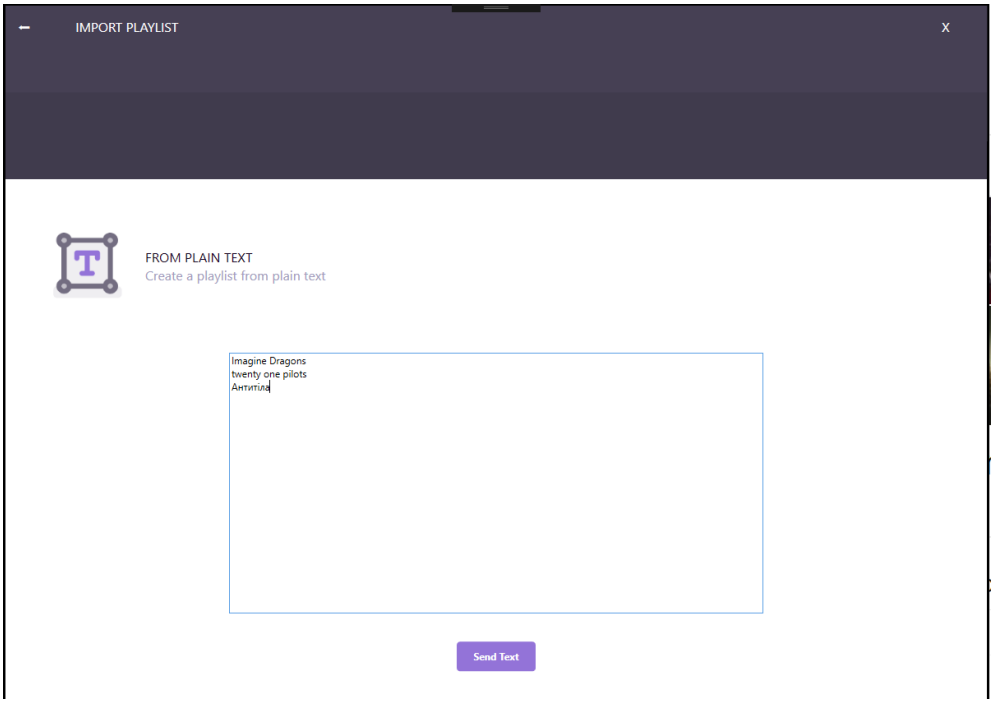


Рисунок 3.21 – Введення музичних груп для імпорту

Далі йде вікно, яке дозволить користувачеві обрати всі необхідні треки для подальшого знайдення і добавлення в список відтворення рис 3.22

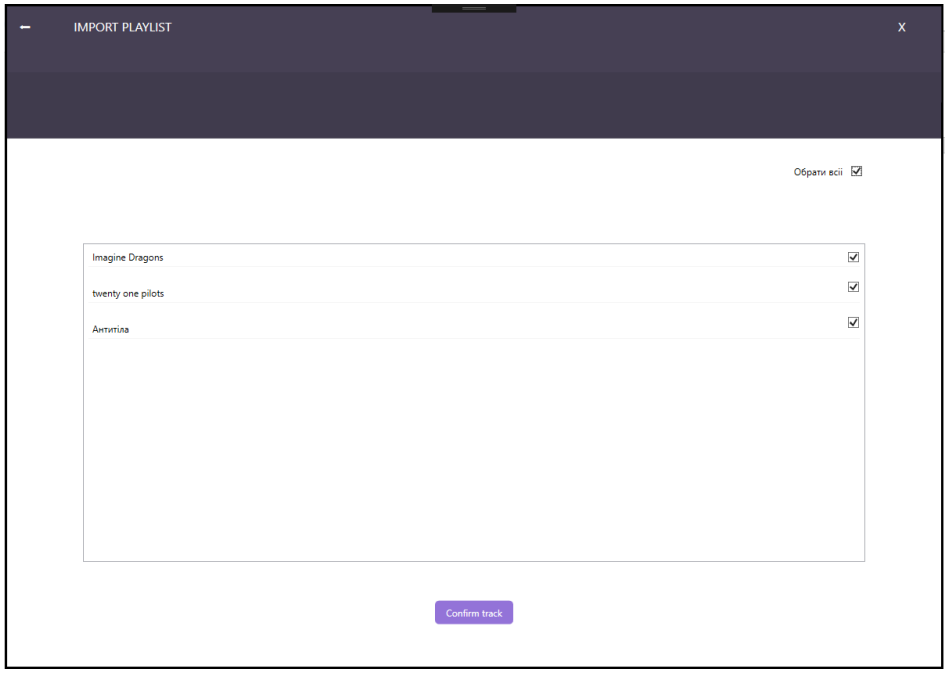


Рисунок 3.22 – Вибір треків для подальшого імпорту

Далі йде вікно для введення назви списку відтворення, опису і функції «видалення дублюючих треків» рис 3.23

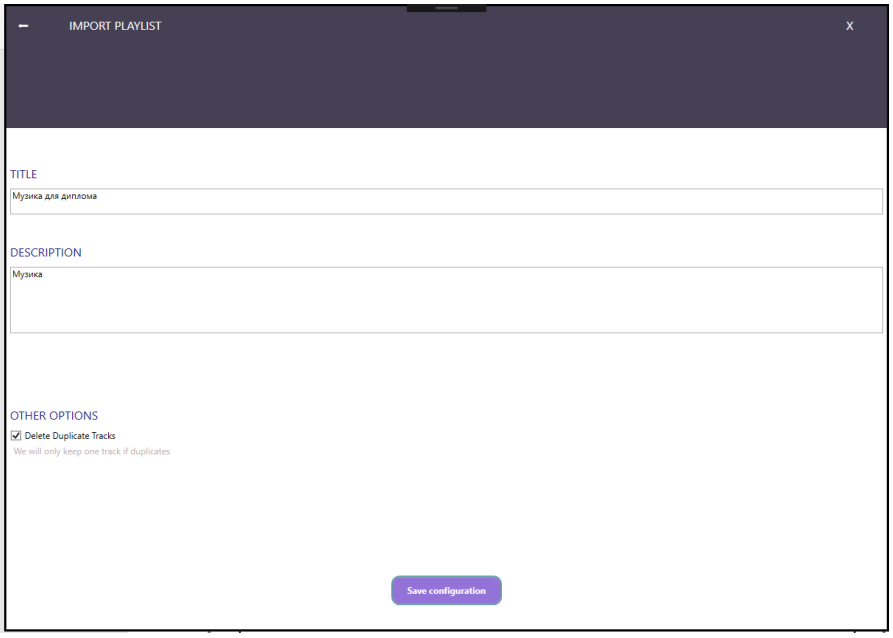


Рисунок 3.23 – Налаштування списку відтворення

Після налаштування списку відтворення йде вікно вибору музичного сервісу рис 3.24

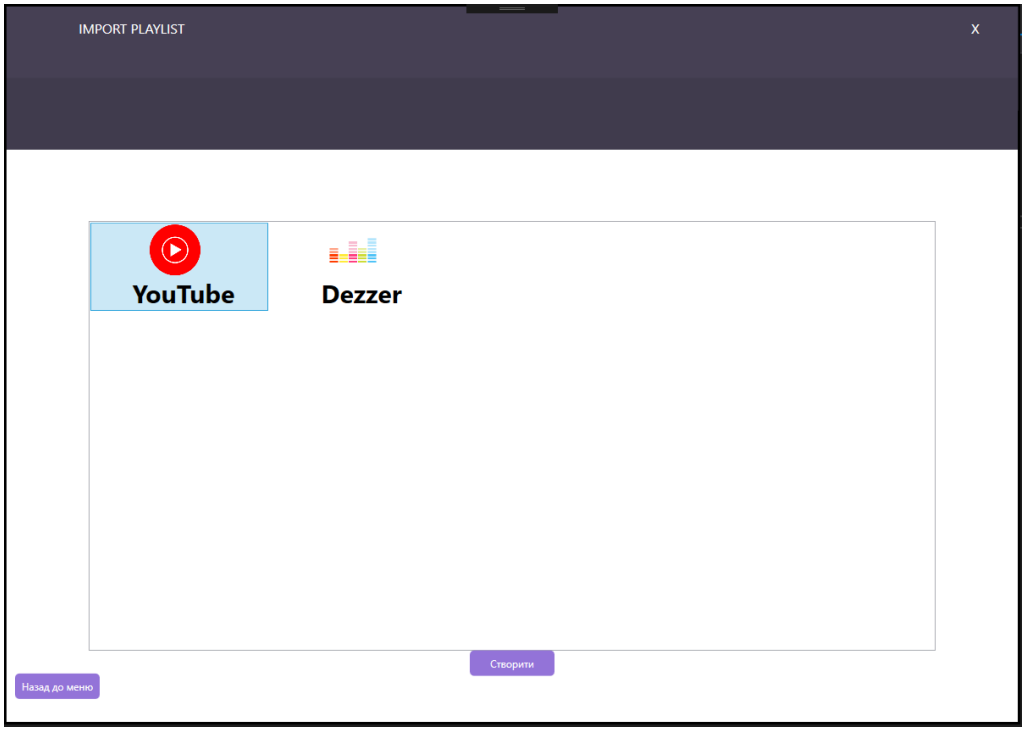


Рисунок 3.24 – вибір музичного сервісу

Далі відповідно до обраного музичного сервісу після натискання кнопки «Створити» відправляється запит на шину даних з токеном доступу, після чого даний запит перенаправляється до мікросервісу YouTube Music. Даний мікросервіс обробляэ дані, і за допомогою токена доступу і роботою із зовнішніми API знаходить відповідні треки, створює список відтворення і повертає відповідь у форматі JSON до шини даних, яка, в свою чергу, відправить це користувачеві. Структура JSON виглядатиме наступним чином рис 3.25


```
{
   "data":[
      {
         "id":"0I647GU3Jsc",
         "artistName":null,
         "album":null,
         "duration":null,
         "name":"Imagine Dragons - Natural"
      },
      {
         "id":"1oOWKm8GW6A",
         "artistName":null,
         "album":null,
         "duration":null,
         "name":"twenty one pilots - Level of Concern (Official Video)"
      },
      {
         "id":"0kExxJnXbQE",
         "artistName":null,
         "album":null,
         "duration":null,
         "name":"Антитіла - Вірила / Official Video"
      }
   ],
   "success":true,
   "message":null
}
```

Рисунок 3.25 – Результат створення списку відтворення

Консоль роботи сервера шини даних і серевера YouTubeMusic виглядає наступним чином рис 3.26, 3.27

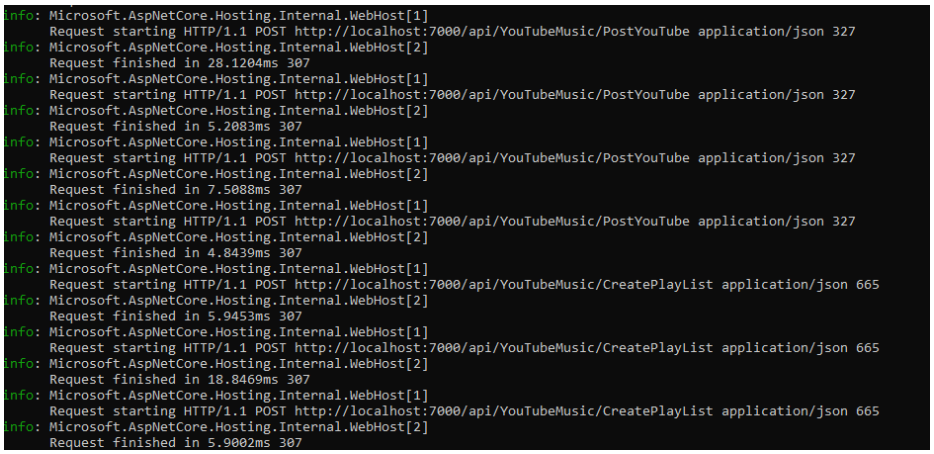


Рисунок 3.26 – Результат створення списку відтворення робота шини даних

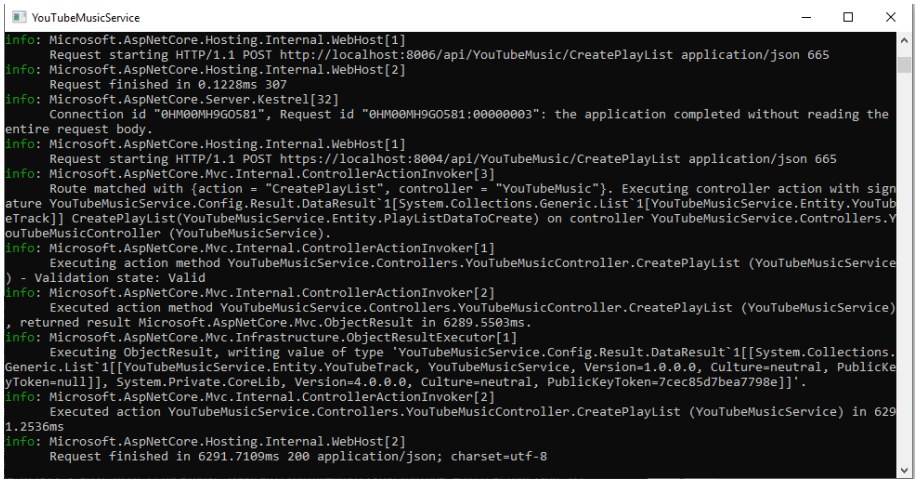


Рисунок 3.27 – Результат створення списку відтворення робота YouTubeMusicService

ВИСНОВКИ

Під час виконання дипломного проєкту було поставлено завдання централізованого управління і агрегації музичних сервісів. Було оглянуто і проаналізовано готові існуючі рішення: Musconv, Soundizz. Після огляду було виявлено ряд переваг даних платформ, які були пов’язані із зручністю створення нових списків відтворення за допомогою програмної обробки файлів спеціалізованого розширення, можливістю створювати список відтворення за допомогою URL – посилання, а також введення простого тексту користувачем. Також було виявлено недоліки, які пов’язані із платною підпискою для користувачів, накладання обмежень по створенню нових списків, а також «зависання» додатку в період обробки великої кількості даних.

Враховуючи нагруженість по обробці даних для даного класу підсистем було спроектовано архітектурне рішення, яке допоможе рівномірно розприділити навантаження. Для виконання даного завдання найоптимальнішим варіантом являється клієнт – серверна архітектура, серверна частина якої будуватиметься на основі мікросервісів. Для рівномірного розподілення навантаженості системи було прийнято рішення для кожного із музичних сервісів реалізувати окремий додаток(мікросервіс), який буде працювати незалежно від інших і оброблятиме інформацію лише для одного музичного сервісу.

Для реалізації технічного завдання в рамках змодельованої архітектури було оглянуто і проаналізовано технології для створення серверних частини додатку: Spring (Java), ASP.NET Core (C#) , GGI (C++). В ході порівняння для реалізації серверної частини додатку було обрано технологію ASP.NET Core як найоптимальнішу по швидкодії серреалізації структур даних, а також швидкості розробки. В межах порівняння технологій по реалізації клієнтської частини додатку було оглянуто технології Swing, WPF, Windows Forms. Для реалізації клієнтської частини додатку було обрано технологію WPF, як найбільш оптимальну по критеріям швидкодії запуску і відображення графічних об’єктів.

Після вивчення документації API по роботі із музичними сервісами було обрано сервіси YouTube Music і Deezer для інтеграції до обумовленої системи. Для

обробки запитів для кожної платформи був реалізований окремий мікросервіс і шина даних, яка маршрутизувала мережеві пакети між сервісами і клієнтом.

В результаті було реалізовано зручний графічний віконний додаток користувача, який поєднує переваги своїх аналогів, та не допускає недоліків по завданню централізованого управління і агрегації музичних сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Конвертор списків відтворення Soundiiz URL: https://soundiiz.com/
2. Конвертор списків відтворення musconv URL: https://musconv.com/
3. Клієнт – серверна архітектура додатків URL: https://www.britannica.com/technology/client-server-architecture#:~:text=Client%2Dserver%20architecture%2C%20architecture%20of,the%20results%20the%20server%20returns.
4. Сервіс – орієнтована архітектура додатків URL: https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec
5. Архітектура розподілених систем URL: https://link.springer.com/chapter/10.1007/978-1-4302-0860-0_18
6. Мікросервісна архітектура URL: https://www.pluralsight.com/courses/microservices-architecture?aid=7010a000002LUv2AAG&promo=&utm_source=non_branded&utm_medium=digital_paid_search_google&utm_campaign=XYZ_EMEA_Dynamic&utm_content=&gclid=CjwKCAjw5vz2BRAtEiwAbcVILy33-8gSMHuLU4F5zoa35iQxJehOnGC1uJ6wzXipd7ctH3XbzNxhOhoCpLEQAvD_BwE
7. Протокол передачі даних gRPC URL: https://grpc.io/
8. Протокол передачі даних HTTP URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
9. Протокол передачі даних AMQP URL: https://www.amqp.org/
10. Інтерфейс програмування додатків URL: https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/
11. Захищений протокол авторизації OAuth2.0 URL: https://oauth.net/2/

12. Маркер доступу для протокола авторизації :https://auth0.com/docs/tokens/concepts/access-tokens
13. Порівняння Java і .NET: https://www.codeproject.com/Articles/92812/Benchmark-start-up-and-system-performance-for-Net
14. Введення в Win32 API URL: http://cppstudio.com/post/9384/#:~:text=Win32%20API%20(%D0%B4%D0%B0%D0%BB%D0%B5%D0%B5%20WinAPI)%20%E2%80%93,%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D1%8E%D1%89%D0%B8%D1%85%20%D0%BF%D0%BE%D0%B4%20%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%D0%BC%20%D0%9E%D0%A1%20Windows.
15. Порівняння технологій WPF Windows Forms URL: https://www.grapecity.com/blogs/flexgrid-performance-compare-desktop-platforms
16. Java Server Pages URL: https://metanit.com/java/javaee/3.1.php
17. Опис шаблону MVC URL: https://www.techopedia.com/definition/27454/model-mvc-aspnet
18. Потоки виведення, OutputStream URL: https://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html
19. Порівнюючи технології GGI та ASP.NET Core URL: https://ela.kpi.ua/bitstream/123456789/27860/1/Vovk_magistr.pdf
20. Порівняння техгології, які написані на базі Spring і ASP.NET Core URL: https://www.techempower.com/benchmarks/
21. Протокол передачі даних HTTPS URL: https://www.tutorialsteacher.com/https/what-is-https
22. Модель відкритих систем OSI URL: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/

23. Принцип роботи Ocelot на базі .NET Core URL: https://ocelot.readthedocs.io/en/latest/introduction/gettingstarted.html
24. Опис документації API YouTube Music URL: https://developers.google.com/youtube/v3
25. Опис документації API Deezer URL:https://developers.deezer.com/api/explorer
26. Бібліотека по роботі із сервісами Google URL: https://developers.google.com/api-client-library/dotnet/apis/youtube/v3
27. Структура відповіді для сервіса Deezer URL: https://developers.deezer.com/api
28. Подання кирилиці в UTF-8 URL: https://i.voenmeh.ru/kafi5/Kam.loc/inform/UTF-8.htm
29. Покрокове керівництво. Створення кнопки за допомогою Microsoft Expression Blend URL: https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/controls/walkthrough-create-a-button-by-using-microsoft-expression-blend
30. Програми, що використовують CEF URL: https://en.wikipedia.org/wiki/Chromium_Embedded_Framework
31. Дослідження Dependency Injectiony контейнерах C URL: https://rubikscode.net/2018/04/09/exploring-dependency-injection-in-c-and-top-3-di-containers-part-2/
32. Шаблон абстрактна фабрика URL: https://devpractice.ru/patterns-abstract-factory/